z/OS

Text Search: Programming the Text Search Engine

Version 1.2

SH12-6717-01

z/OS

Text Search: Programming the Text Search Engine

Version 1.2

SH12-6717-01

Note!

Before using this information and the product it supports, be sure to read the general information under “Appendix C. Notices” on page 219.

Second Edition, October 2001

This edition applies to Version 1.2 of z/OS (Program Number 5694-A01) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this book vii
Who should read this book vii
The Text Search Library. vii

Chapter 1. Before you start 1
Overview of the interfaces 1
The Text Search Engine application programming interface 1
The Text Search Engine library service interfaces 1
The Text Search Engine text format. 1
Data exchange conventions 1
Basics about the data streams 2
Passing numerical values 3
Passing character data 3

Chapter 2. Linguistic processing 5
Linguistic processing for indexing 5
Basic text analysis 6
Reducing terms to their base form 7
Stop-word filtering 7
Decomposition (splitting compound terms) 7
Linguistic processing for retrieval. 8
Synonyms 8
Thesaurus expansion 9
Sound expansion 9
Character and word masking 9
The supported languages 9
Thesaurus concepts 10
Terms 11
Relations 11
Ngram thesaurus relations. 13
Creating a Text Search Engine thesaurus 13
Creating an Ngram thesaurus 16

Chapter 3. Using the API functions. 21
Listing the servers and starting a session 21
Creating and setting up an index 21
Creating and deleting an index 22
Getting the names of the existing indexes 22
Suspending and resuming an index 23
Opening an index 23
Getting index information 23
Setting and getting the default indexing rules 23
Working with an index 24
Scheduling and indexing documents 24
Getting the index status 25
Improving performance and the use of storage 25
Setting and getting the index function status 26
Searching for documents 26
The search functions. 26
Specifying the scope of a cross-index search. 27
Working with the result of a search 28
Finding matches in search results 28
Getting problem information for a cross-index search 29

Working with result views 29
Creating and ranking a result view 29
Selecting from and sorting a result view 30
Getting the contents of a result view 31
Working with structured documents 31

Chapter 4. Setting up your application 33
What the API provides 33
Environment and system overview 33
Setting up your application 34
Defining symbolic names 34
Linking your application 34
Some programming hints 35

Chapter 5. Calling the API functions 37
EhwAddQueryScope 38
EhwCancelContinuation 41
EhwClearIndex 43
EhwClearScheduledDocuments 45
EhwCloseDocument 47
EhwCloseIndex 48
EhwCreateDocumentModel 49
EhwCreateIndex 52
EhwCreateIndexGroup 58
EhwCreateResultView 60
EhwDeleteDocumentModel 62
EhwDeleteIndex 64
EhwDeleteIndexGroup 66
EhwDeleteResult 67
EhwDeleteResultView 69
EhwDelIndexingMsgs 70
EhwEndSession 71
EhwGetDocumentModel 72
EhwGetIndexFunctionStatus 75
EhwGetIndexInfo 78
EhwGetIndexingMsgs 81
EhwGetIndexingRules 84
EhwGetIndexStatus 87
EhwGetMatches 90
EhwGetProblemInfo 94
EhwGetResultView 96
EhwListDocumentModels 99
EhwListIndexes 102
EhwListResult 104
EhwListServers 107
EhwOpenDocument 110
EhwOpenIndex 113
EhwRank 115
EhwReorgIndex 118
EhwResumeIndex 120
EhwScheduleDocument 122
EhwSearch 125
EhwSelectResultView 139
EhwSetIndexFunctionStatus 142
EhwSetIndexingRules 144
EhwSort 147

EhwStartSession 150
EhwSuspendIndex 154
EhwUpdateIndex 156

Chapter 6. Connecting Text Search Engine to a library 159
What library services provide 159
Functions 160
Setting up the library connection 160

Chapter 7. Specifications for library services 163
LIB_access_doc 164
LIB_close_doc 166
LIB_doc_index_status 167
LIB_end 169
LIB_get_doc_attr_values 170
LIB_get_doc_group_attr_values 173
LIB_init 176
LIB_list_doc_groups 178
LIB_list_documents 181
LIB_read_doc_content 184
Error handling concept and return codes 185

Chapter 8. Using the Text Search Engine text format 187
Data stream syntax 187

Appendix A. The API return codes 191
API error handling 191
Types of return codes 191

Appendix B. Error codes returned by GetIndexingMsgs and GetIndexFunctionStatus 207

Appendix C. Notices 219
Programming interface information 220
Trademarks 220

Index 221

About this book

This book describes the following interfaces:

- The Text Search Engine application programming interface (API)
- The Text Search Engine library service interfaces
- The Text Search Engine text format

It explains how these interfaces can be used to:

- Obtain Text Search Engine services from customer applications
- Provide library services (document access) to Text Search Engine
- Support additional text formats in Text Search Engine and exploit Text Search Engine support for multilingual documents

Who should read this book

This book contains information for application programmers who want to integrate Text Search Engine functions with customer applications, document repositories (libraries), or multimedia display services. It is written for application programmers who want to develop an application that can use the services of the Text Search Engine functions, or who want to develop programs that enable Text Search Engine to interface with a given document repository. All programming interfaces provided by Text Search Engine are for use with the C programming language. The Text Search Engine text format is a data interchange format, not a programming interface.

Before using the interface descriptions in this book, you should read “Chapter 1. Before you start” on page 1. It provides an overview of the Text Search Engine interfaces, defines most of the interface concepts and terminology, and explains the notation conventions used in this book.

The Text Search Library

The following books are available for z/OS Text Search:

- z/OS Text Search: Installation and Administration for the Text Search Engine, SH12-6716
- z/OS Text Search: Programming the Text Search Engine, SH12-6717
- z/OS Text Search: NetQuestion Solution, SH12-6718

Chapter 1. Before you start

This chapter provides an overview of the Text Search Engine interfaces described in subsequent chapters, defines most of the interface concepts and terminology, and explains the notation conventions used in this book.

Overview of the interfaces

The Text Search Engine application programming interface

The Text Search Engine API can be used as an interface between application programs and Text Search Engine functions. It provides a general-purpose interface for obtaining the information retrieval services of Text Search Engine, such as searching for information or having documents indexed.

Application programs using the Text Search Engine API must be written in the C programming language. Each Text Search Engine API function is invoked with a CALL statement (C function call). All arguments are passed to Text Search Engine separately as input parameters and results are returned as output parameters. The conventions for data exchange and syntax notation are described in “Data exchange conventions”.

The Text Search Engine library service interfaces

The library service interfaces are used as an interface between Text Search Engine and the library system (or document access method) that manages the documents indexed by Text Search Engine. They constitute a set of general-purpose interfaces to provide library services to Text Search Engine, such as accessing (reading) the documents to be indexed.

Text Search Engine provides a set of these service interfaces for flat files. Installations can provide additional sets to support their own library system.

Each Text Search Engine library service function is invoked with a CALL statement (C function call). The conventions for parameter passing and syntax notation are described in “Data exchange conventions”.

The Text Search Engine text format

The Text Search Engine text format is a document format tailored to the needs of indexing natural-language text. Installations or applications can pass documents in this format to the Text Search Engine indexing services.

If you need support for multilingual documents, you can use the Text Search Engine text format which allows you to switch languages within a document.

You should consider using the Text Search Engine text format as an interchange format if you need to convert your documents into a format supported by Text Search Engine. The document formats supported by Text Search Engine are listed in file IMOLSDEF.H of the library services toolkit.

Data exchange conventions

All Text Search Engine programming interfaces are call interfaces using C function calls. Data exchange is done explicitly via parameters of these functions.

In the Text Search Engine API and library service interfaces, there are two types of parameters:

- **Simple parameters**

  These are single values of a basic data type and include pointers, integers, and characters. Specification rules for these parameters are given in “Passing numerical values” on page 3 and “Passing character data” on page 3.

- **Data stream parameters**

  These contain information of varying length and content. See “Basics about the data streams” for an explanation of Text Search Engine data streams. The length of a data stream parameter is passed as a separate integer parameter.

The Text Search Engine text format is a data stream format that conforms to the concepts of Text Search Engine data streams explained in “Basics about the data streams”.

Basics about the data streams

Data streams are a way of passing information of variable length and content. A data stream is a sequence of information pieces, called **data stream items**, that:

- Identify themselves
- Have a common format so that they can be parsed by a program that does not understand their purpose or information content.

This also allows programs to select only the information they need.

The data stream items are split into different fields. Each field, its length, and a description of the data that it contains, is given in Table 1.

Table 1. Format of data stream items

Field	Length	Description
ll	2 bytes	Total length of the item, including this length field. This field is in big-endian format; that is: value-of-ll = first byte × 256 + second byte.
id	2 bytes	Item identifier. Each identifier has a mnemonic code (such as IRS or TERM) that is used throughout this book. The actual 2-byte numeric values are defined in the toolkit files supplied with Text Search Engine. This field is in big-endian format; that is: value-of-id = first byte × 256 + second byte.
t	1 byte	Type of data stream item. There are three types of data stream item: Start, End, and Atomic. Start and end items (with identical identifiers) are used to delimit related data stream items. If a program ignores a start item, it should also ignore all subsequent items up to, and including, the matching end item. Atomic items contain the actual information pieces. Throughout this book, the three types are shown as S, E, and A, respectively.

Table 1. Format of data stream items (continued)

Field	Length	Description
data	variable	Data passed with the item. This field is optional. Only the atomic data stream items contain a data field. The format of item data is either explicitly defined for the particular item or governed by the rules outlined in “Passing numerical values” and “Passing character data”.

Throughout this book, data stream items are represented as:

```
ll id t data
```

The blank spaces are shown to improve readability; they are not part of the actual data stream.

The syntax of data stream parameters and the syntax of the Text Search Engine text format, a data stream format for documents, are specified in syntax diagrams.

Passing numerical values

Numerical values, passed as simple parameters, are always in the standard local representation. On Intel-based platforms, this is the *little-endian* format, where bytes are swapped in binary representation. For example, the value of a two-byte unsigned integer is equal to:

```
(second byte × 256) + first byte
```

As a programmer, you need not be concerned about this because this representation is handled properly by your system.

Data streams are designed to be independent of local conventions so that they can be passed across mixed environments. Text Search Engine uses the *big-endian* format for all numerical values in Text Search Engine data streams. This is the format where bytes are not swapped in binary representation. Thus, the value of a two-byte unsigned integer is equal to:

```
(first byte × 256) + second byte
```

As a programmer on Intel-based platforms, you need to be aware of this because it is not the standard local representation.

To facilitate handling big-endian values on Intel-based platforms, the Text Search Engine toolkits contain macros that convert big-endian format to little-endian, and vice versa. The programming hints and samples for the individual interfaces contain more details about these conversion macros.

In the library service interfaces, some data streams contain fields with date-time values. The format of these values is explicitly defined in the Text Search Engine library services toolkit.

Passing character data

Whenever character data is passed in interfaces, especially across systems or environments, character encoding problems arise. Text Search Engine avoids these problems whenever possible and sets up strict and simple rules when these problems cannot be avoided.

The following rules apply to character data in parameters of the API and the library service interfaces.

In the API functions, only data stream parameters can contain character data. (Simple parameters defined as **char** actually contain binary information with no code page dependencies.) In the library service functions, character data is sometimes passed in simple parameters.

There are five types of character data:

- **Text Search Engine names** (such as index names or server names)

  These are always passed in coded character set 00819 (ISO 8859-1) representation. Since Text Search Engine uses a very limited character set (uppercase letters A-Z, digits 0-9) for names, there should usually be no transformation required for these names within your application or library functions.
- **External names** (such as document names)

  These are always passed in the local system code page, because they are entered by a user on a local workstation, or are ready to be displayed on the screen of a local workstation.
- **Document identifiers** (and document group identifiers)

  These are not considered to be character data by Text Search Engine. There is no coded character set associated with these identifiers. Instead, they are handled as bit strings and are not transformed even when passed across systems or environments.
- **Document text**

  The textual content of documents to be indexed is always associated with a coded character set and language, either explicitly (for example, when you use the Text Search Engine text format) or implicitly by using defaults set by the administrator.
- **Search terms**

  These are always accompanied by a CCSID (coded character set identifier) and language code so they can be properly handled by any Text Search Engine server.

Chapter 2. Linguistic processing

Text Search Engine offers linguistic processing in these areas:

- **Indexing**. When Text Search Engine analyzes documents to extract the terms to be stored in the text index, the text is processed linguistically to extract significant terms for the index. This is done to make retrieval as simple and as fast as possible.
- **Retrieval**. When Text Search Engine searches through the document index to find the names of documents that contain occurrences of the search terms you have specified, the search terms are also processed linguistically to match them with the indexed terms.
- No linguistic processing is applied to Ngram indexes.

The terms in an Ngram index are stored as they occur in the documents; no linguistic processing is applied.

Linguistic processing for indexing

When Text Search Engine indexes and retrieves documents, it makes a linguistic analysis of the text.

The linguistic processing used for indexing documents consists of:

- Basic text analysis
  - Recognizing terms (tokenization)
  - Normalizing terms to a standard form
  - Recognizing sentences
- Reducing terms to their base form
- Stop-word filtering
- Decomposition (splitting compound terms).

This table shows a summary of how terms are indexed when the index type is **linguistic** and no additional index properties have been requested.

Table 2. Term extraction for a linguistic index

Document text	Term in index	Linguistic processing
Mouse Käfer	mouse kaefer	Basic text analysis (normalization)
mice swum	mouse swim	Reduction to base form
system-based Wetterbericht	system-based system base wetterbericht wetter bericht	Decomposition
a report on animals	report animal	Stop-word filtering. Stop words are: a, on

By comparison, the following table shows a summary of how terms are indexed when the index type is **precise.**

Table 3. Term extraction for a precise index

Document text	Term in index	Linguistic processing
Mouse Käfer	Mouse Käfer	No normalization
mice swum	mice swum	No reduction to base form
a report on animals	report animals	Stop-word filtering. Stop words are: a, on
system-based Wetterbericht	system-based Wetterbericht	No decomposition
Sage mir	Sage sage	Sentence-begin processing

Basic text analysis

Text Search Engine processes basic text analysis without using an electronic dictionary.

Recognizing terms that contain nonalphanumeric characters

When documents are indexed, terms are recognized even when they contain nonalphanumeric characters, for example: “don’t”, “$14,225.23”, “mother-in-law”, and “10/22/90”.

The following are regarded as part of a term:

- Accents and apostrophes
- Currency signs
- Number separator characters (like “/” or “·”)
- The “@” character in e-mail addresses (English only).

Language-specific rules are also used to recognize terms containing:

- Accented prefixes in Roman languages, such as `l'aventure` in French.
- National formats for dates, time, and numbers.
- Alternatives, such as `mission/responsibility`, indicated in English using the “/” character.
- Trailing apostrophes in Italian words like `securita'`. It is usual in typed Italian text, when the character set does not include characters with accents, to type the accent *after* the character; for example, “à” is typed “a'”.

Normalizing terms to a standard form

Normalizing reduces mixed-case terms, and terms containing accented or special characters, to a standard form. This is done by default when the index type is linguistic. In a precise index the case of letters is left unchanged; searches are case-sensitive.

For example, the term `Computer` is indexed as `computer`, the uppercase letter is changed to lowercase. A search for the term `computer` finds occurrences not only of `computer`, but also of `Computer`. The effect of normalization during indexing is that terms are indexed in the same way, regardless of how they are capitalized in the document.

Normalization is applied not only during indexing, but also during retrieval. Uppercase characters in a search term are changed to lowercase before the search is made. When your search term is, for example, `Computer`, the term used in the

search is computer. The combined effect of using normalization during indexing and during retrieval is that, regardless of how you capitalize a search term, and regardless of how it is capitalized in the document, the term is found.

Accented and special characters are normalized in a similar way. Any variation of the French word école, such as École, finds école, Ecole, and so on. The German word Bürger finds buerger, Maße finds masse.

If the search term includes masking (wildcard) characters, normalization is done before the masking characters are processed. Example: Bür_er becomes buer_er.

Recognizing sentences

You can search for terms that occur in the same sentence. To make this possible, each document is analyzed during indexing to find out where each sentence ends.

Reducing terms to their base form

In a linguistic index, you can search for mouse, for example, and find mice. Terms are reduced to their base form for indexing; the term mice is indexed as mouse. Later, when you use the search term mouse, the document is found. The document is found also if you search for mice.

The effect is that you find documents containing information about mice, regardless of which variation of the term mouse occurs in the document, or is used as a search term.

In the same way, conjugated verbs are reduced to their infinitive; bought, for example, becomes buy.

Stop-word filtering

Stop words are words such as prepositions and pronouns that occur frequently in documents, and are therefore not suitable as search terms. Such words are in a stop-word list associated with each dictionary, and are excluded from the indexing process.

Stop word processing is case insensitive. So a stop word about also excludes the first word in a sentence About. Stop-word filtering is in effect only when the stop word list for the language in question is in the resource directory.

Decomposition (splitting compound terms)

German is rich in compound terms, like Versandetiketten, which means mail (Versand) labels (Etiketten). Such compound terms can be split into their components.

For a precise index, compound terms are indexed unchanged as one word. For a linguistic index, compound terms are split during indexing. When you search, compound terms are split if you have a linguistic index.

The components are found if they occur in any sequence in a document as long as they are contained within one sentence. For example, when searching for the German word Wetterbericht (weather report), a document containing the phrase Bericht über das Wetter (report about the weather) would also be found.

An attempt is made to split a term if:

- The term has a certain minimum length

- The term is not itself an entry in the electronic dictionary—compounds that are commonly used like the German word `Geschäftsbericht` (business report) *are* in the German dictionary.

If a split is found to be possible, the term's component parts are then reduced to their base form. For example, the compound term `Kindersprachen` has the component parts `kindersprache`, `kind`, and `sprache`.

Linguistic processing for retrieval

Linguistic processing of search terms is controlled by the user through qualifiers in the Text Search Engine query language. "EhwSearch" on page 125 explains the effect of those qualifiers and relates them to the linguistic indexing functionality.

Query processing aims at making search terms weaker so that the recall rate of searches is increased, that is, more relevant documents are found. There are two basic operations on query terms to achieve that goal; they are expansions and reductions. In addition, some search term operations involve both expansion and reduction.

- Expansions take a word or a multi-word term from within a search term and associate it with a set of alternative search terms, each of which may be a multi-word term itself. The source expression and the set of target expressions form a Boolean OR-expression in Text Search Engine's query language. As expansions leave the source term unchanged, they are to some extent independent of the index type. The following are expansion operations:
  - Synonym expansion
  - Thesaurus expansion
- Reductions change the search term to a form that is more general than the one specified by the user. Because it changes the search term, reductions are dependent on the index type to ensure that the changed term matches. Therefore, Text Search Engine derives reduction information from the type of those indices or index that the query is directed against. The following are reductions:
  - Lemmatization
  - Normalization
  - Stop words.
- Some operations both change the search term and expand it with a set of alternative terms. Due to the inherent reduction, these again depend on information contained in the index.

  The following operations fall into this class:
  - Character and word masking
  - Sound expansion.

Synonyms

Synonyms are semantically related words. Usually, these words have the same word class or classes (such as noun, verb, and so on) as the source term. Synonyms are obtained from a lexical resource which is specific for each language. They are always returned in base form and, up to a few exceptions, are not multi-word terms. Search term words are always reduced to their base form when looking up synonyms. Synonym expansion cannot be applied to Ngram indexes.

Here are some examples of synonyms:

- English

```
word - comment remark statement utterance term expression
communication message assurance guarantee warrant bidding command
charge commandment dictate direction directive injunction instruction
mandate order news advice intelligence tidings gossip buzz cry
hearsay murmur report rumor scuttlebutt tattle tittle-tattle
whispering
```

- French

```
mot - expression parole terme vocable lettre billet missive epître
plaisanterie
```

- German

```
Wort - Vokabel Bezeichnung Benennung Ausdruck Begriff Terminus
Ehrenwort Brocken Bekräftigung Versprechen Zusicherung Gelöbnis
Beteuerung Manneswort Schwur Eid Ausspruch
```

Thesaurus expansion

A search term can be expanded using thesaurus terms that can be reached through a specific relation. These relations can be hierarchical (such as the “Narrower term” relation), associative (such as a “Related term” relationship), or it can be a synonym relationship. A thesaurus term can be, and often is, a multi-word term.

“Thesaurus concepts” on page 10 describes thesaurus expansion in more detail.

Sound expansion

Sound expansion expands single words through a set of similarly sounding words. It is particularly useful whenever the exact spelling of a term to be searched is not known. Sound expansion cannot be applied to Ngram indexes.

Character and word masking

Masking is a non-linguistic expansion technique, where a regular expression is replaced with the disjunction of all indexed words that satisfy it. Neither a masked expression nor any of its expansions is subject to lemmatization, stop-word extraction, or any of the other expansion techniques. This can have the effect that, for example, an irregular verb form like `swum`, when searched with the masked term `swu*`, is matched on a precise index, but not on a linguistic index, where this form has been lemmatized to become `swim`.

The supported languages

The following list shows the SBCS document languages supported by Text Search Engine. Decomposition is supported for German only. The feature index, and functions based on it, are supported for US and UK English only. Synonyms are not supported for Norwegian Nynorsk and Bokmal. Only the logical (not the visual) file format for documents written in bidirectional languages is supported.

Arabic
Brazilian
Canadian French
Catalan
Danish
Dutch
Finnish
French
German
Hebrew
Icelandic
Italian

Norwegian Bokmal
Norwegian Nynorsk
Norwegian Bokmal and Nynorsk
Portuguese
Russian
Spanish
Swedish
Swiss German
UK English
US English

Thesaurus concepts

A thesaurus is a controlled vocabulary of semantically related terms that usually covers a specific subject area. It can be visualized as a semantic network where each term is represented by a node. If two terms are related to each other, their nodes are connected by a link labeled with the relation name. All terms that are directly related to a given term can be reached by following all connections that leave its node. Further related terms can be reached by iteratively following all connections leaving the nodes reached in the previous step. Figure 1 shows an example of the structure of a very small thesaurus.

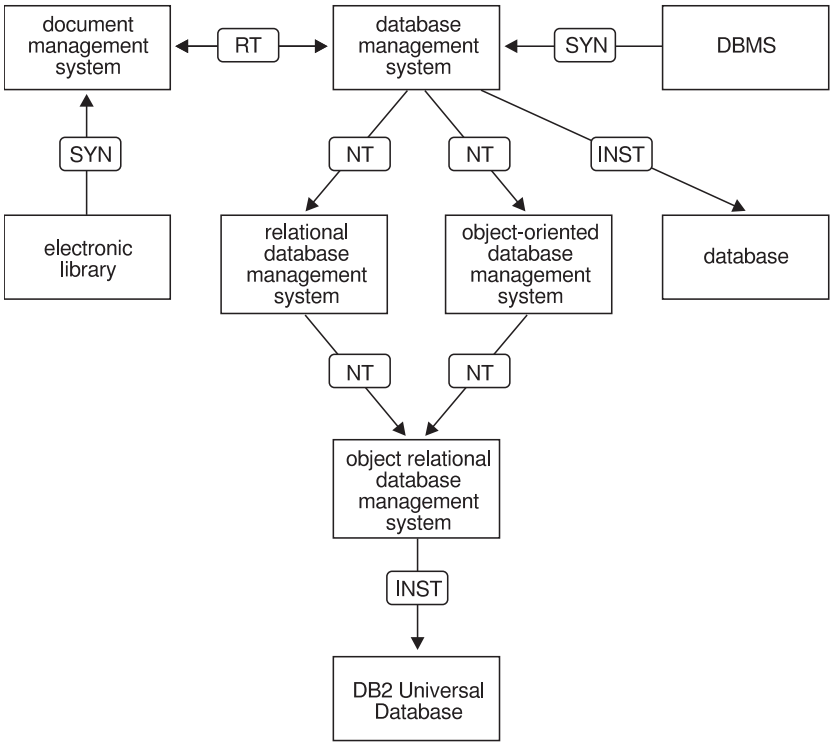


Figure 1. A thesaurus displayed as a network

Text Search Engine lets you expand a search term by adding additional terms from a thesaurus that you have previously created. Refer to “EhwSearch” on page 125 to find out how to use thesaurus expansion in a query.

To create a thesaurus for using it in a search application requires a thesaurus definition file that has to be compiled into an internal format, the thesaurus dictionary.

The dictionary format used by an index of type XTYP_LINGUISTIC and XTYP_PRECISE differs from the one used by an Ngram index. Thus two different thesaurus compilers are provided with the product. They are not only slightly different in the concepts they are based on, but require different source formats. So you should first decide which index type you will use before you start defining the thesauri for your search application.

A thesaurus that can be used with an index of type XTYP_PRECISE or XTYP_LINGUISTIC is referred to as a Text Search Engine thesaurus. A thesaurus for an Ngram index is called an Ngram thesaurus.

The basic components of a Text Search Engine thesaurus or an Ngram thesaurus are “terms” and “relations”.

Terms

A term is a word or expression denoting a concept within the subject domain of the thesaurus. For example, the following could be terms in one or more thesauri:

```
data processing
helicopter
gross national product
```

In a Text Search Engine thesarus, terms are classified as either descriptors or nondescriptors. A *descriptor* is a term in a class of synonyms that is the preferred term for indexing and searching. The other terms in the class are called *nondescriptors*. For example, `outline` and `shape` are synonymous, where `shape` could be the descriptor and `outline` a nondescriptor.

An Ngram thesaurus does not distinguish between descriptors and nondescriptors.

Relations

A relation is an expression of an association between two terms. Relations have the following properties:

- The *depth* of a relation is the number of levels over which the relation extends. This is specified in the search syntax using the THESDEPTH keyword. Refer to “EhwSearch” on page 125 to find out how to use thesaurus expansion in a query.
- The *directionality* of a relation specifies whether the relation is true equally from one term to the other (bidirectional), or in one direction only (unidirectional).

Thesaurus expansion can use every relation defined in the thesaurus. You can also specify the depth of the expansion. This is the maximum number of transitions from a source term to a target term. Note however that the term set may increase exponentially as the depth is incremented.

The following example shows those terms that are newly added as the depth increases.

```
health

health service, paramedical, medicine, illness

allergology, virology, veterinary medicine, toxicology, surgery,
stomatology, rhumatology, radiotherapy, psychiatry, preventive
medicine, pathology, odontology, nutrition, nuclear medicine,
neurology, nephrology, medical check up, industrial medicine,
hematology, general medicine, epidemiology, clinical trial,
cardiology, cancerology
```

Text Search Engine thesaurus relations

These are the relation types provided by a Text Search Engine thesaurus:

- Associative
- Synonymous
- Hierarchical
- Other

In a Text Search Engine thesaurus there are no predefined relations. You can give each relation a name, such as BROADER TERM, which can be a mnemonic abbreviation, such as BT. The common relations used in thesaurus design are:

- BT or BROADER TERM
- NT or NARROWER TERM
- RT or RELATED TERM
- SYN or SYNONYM
- USE
- UF or USE FOR

Associative: An associative relation is a bidirectional relation between descriptors, extending to any depth. It binds two terms that are neither equivalent nor hierarchical, yet are semantically associated to such an extent that the link between them may suggest additional terms for use in indexing or retrieval.

Associative relations are commonly designated as RT (related term). Examples are:

dog RT security
pet RT veterinarian

Synonymous: When a distinction is made between descriptors and nondescriptors, as it is in a Text Search Engine thesaurus, the synonymous relation is unidirectional between two terms that have the same or similar meaning. In a class of synonyms, one of the terms is designated as the descriptor. The other terms are then called nondescriptors. Refer to “Ngram thesaurus relations” on page 13 for a definition of the synonymous relation when no distinction is made between descriptors and nondescriptors.

The common designation USE leads from a given nondescriptor to its descriptor. The common designation USE FOR leads from the descriptor to each nondescriptor. For example:

feline USE cat
lawyer UF advocate

Hierarchical: A hierarchical relation is a unidirectional relation between descriptors that states that one of the terms is more specific, or less general, than the other. This difference leads to representation of the terms as a hierarchy, where one term represents a class, and subordinate terms refer to its member parts. For example, the term “mouse” belongs to the class “rodent”.

BROADER TERM and NARROWER TERM are hierarchical relations. For example:

car NT limousine
equine BT horse

Other: A relation of type *other* is the most general. It represents an association that does not easily fall into one of the other categories. A relation of type *other* can be bidirectional or unidirectional, there is no depth restriction, and relations can exist between descriptors and nondescriptors.

This relation is often used for new terms in a thesaurus until the proper relation with other terms can be determined.

Of course you can define your own bidirectional synonymous relation by using the relation type *associative* for a synonymous relation between descriptors or even with the relation type *other* for a synonymous relation between arbitrary terms.

Ngram thesaurus relations

An Ngram thesaurus supports the following two types:

- Associative
- Synonymous

There are two predefined relations, each of them based on one of these two types. You can define your own relations based on the type *associative*. For details, see "Creating an Ngram thesaurus" on page 16.

Associative

An associative relation is a bidirectional relation between two terms that do not express the same concept but relate to each other. The predefined relation *RELATED_TO* and all user-defined relations are based on this relation type.

Examples are:

tennis RELATED_TO racket
German RELATED_TO sausage

Synonymous

A synonym relation is a bidirectional relation between two terms that have the same or similar meaning and can be used as alternatives for each other. This relation can, for example, be used for a term and its abbreviation. The predefined relation SYNONYM_OF is the only relation based on this type.

Examples are:

spot SYNONYM_OF stain
US SYNONYM_OF United States

Creating a Text Search Engine thesaurus

There is a sample English thesaurus compiler input file `imothes.sgm` stored in the resource directory of the installation path. A compiled version of this SGML input is also stored there. The files belonging to this thesaurus are called `imothes.th1`, `imothes.th2`, ..., and `imothes.th6`.

To create a thesaurus, first define its content in a file. It is recommended that you use a plain directory for each thesaurus that you define. The file can have any extension except `th1` to `th6`, which are used for the thesaurus dictionary. If you use the same directory for an Ngram thesaurus, see "Creating an Ngram thesaurus" on page 16 for more excluded file extensions.

Then compile the file by running:

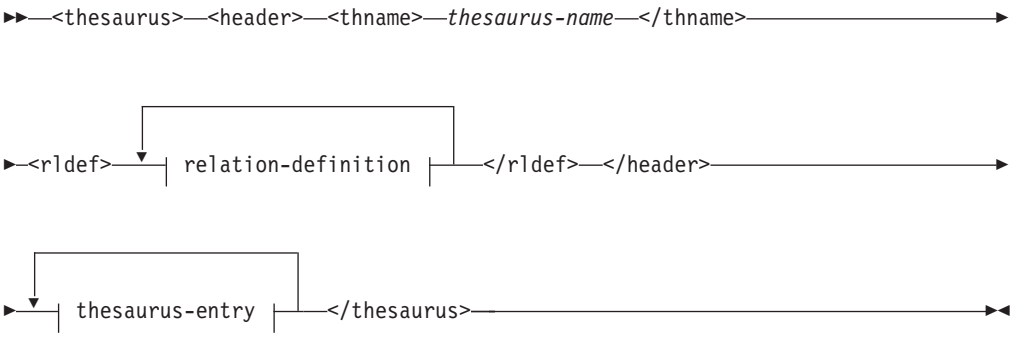
```
imothesc -f filename -ccsid ccsid
```

where *ccsid* is 850 on workstation platforms, or 500 on z/OS.

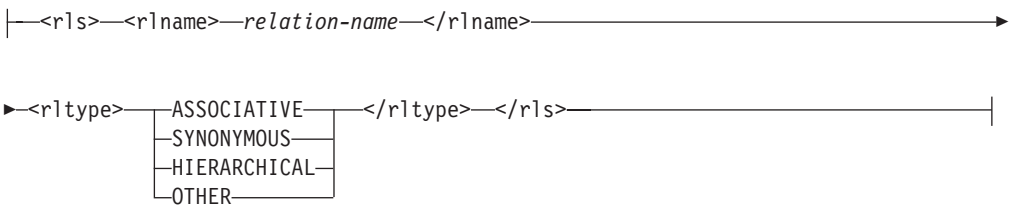
`imothesc` produces thesaurus files having the name *filename* without extension and the extension `th1` to `th6`, in the same directory where the definition file is located. If there is already a thesaurus with the same name, it is overwritten without warning.

Refer to “EhwSearch” on page 125 to find out how to use a thesaurus in a query.

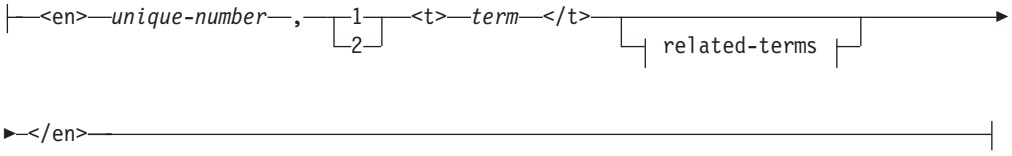
Specify the content of a thesaurus using the Standard Generalized Markup Language (SGML). The following diagram shows the syntax rules to follow when creating a thesaurus.



relation-definition:



thesaurus-entry:



related-terms:

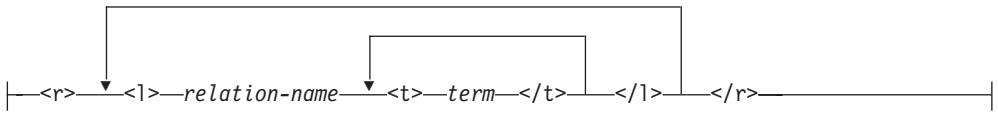


Figure 2 on page 15 shows the SGML definition of the thesaurus shown in Figure 1 on page 10.

```
<thesaurus>
<header>
<thname>thesc example thesaurus</thname>
<rldef>
<rls>
<rlname>Related Term</rlname>
<rltype>associative</rltype>
</rls>
<rls>
<rlname>Narrower Term</rlname>
<rltype>hierarchical</rltype>
</rls>
<rls>
<rlname>Instance</rlname>
<rltype>hierarchical</rltype>
</rls>
<rls>
<rlname>Synonym</rlname>
<rltype>synonymous</rltype>
</rls>
</rldef>
</header>

<en> 2, 1
<t>database management system</t>
<r>
  <l>Narrower Term
  <t>oo database management system</t>
  <t>relational database management system</t>
  </l>
  <l>Synonym
  <t>DBMS</t>
  </l>
  <l>Related Term
  <t>document management system</t>
  </l>
  <l>Instance
  <t>database</t>
  </l>
</r>
</en>
```

Figure 2. The definition of a simple thesaurus (Part 1 of 2)

```
<en> 5, 1
<t> relational database management system </t>
<r>
  <l>Narrower Term
  <t>object relational database management system</t>
  </l>
</r>
</en>

<en> 3, 1
<t>object relational database management system</t>
<r>
  <l>Instance
  <t>DB2 Universal Database</t>
  </l>
</r>
</en>

<en> 6, 1
<t>object oriented database management system</t>
<r>
  <l>Narrower Term
  <t>object relational database management system</t>
  </l>
</r>
</en>

<en> 4, 1
<t>document management system</t>
<r>
  <l>Synonym
  <t>library</t>
  </l>
</r>
</en>

<en> 9, 1
<t>library</t>
</en>

<en> 10, 1
<t>DB2 Unversal Database</t>
</en>

<en> 11, 1
<t>database</t>
</en>
</thesaurus>
```

Figure 2. The definition of a simple thesaurus (Part 2 of 2)

Creating an Ngram thesaurus

There is a sample English Ngram thesaurus compiler input file `imonthes.def` stored in the resource directory of the installation path. A compiled version of this sample thesaurus is also stored there. The files belonging to this thesaurus are called imonthes.<extension> with the following extension where n is a digit:

- For dictionary files: wdf, wdv, grf, grv, MEY, ROS, NEY, SOS, lk*n*
- For temporary files: w*n*f, w*n*v, g*n*f, g*n*v, M!1, M!2, N!1, N!2, R!1, R!2, S!1, S!2, M*nn*, N*nn*, R*nn*, S*nn*, $00, $01, $10, $11, $20, and $21

To create an Ngram thesaurus, first define its content in a definition file. You can have several thesauri in the same directory, but it is recommended that you have a separate directory for each thesaurus. The length of the file name without extension must not exceed 8 characters. The extension is optional but is restricted to 3 characters and should be different from any of the above listed extensions.

If you use the same directory for your Text Search Engine thesauri, do not use the extensions either listed in “Creating a Text Search Engine thesaurus” on page 13 either.

Then compile the file by running:

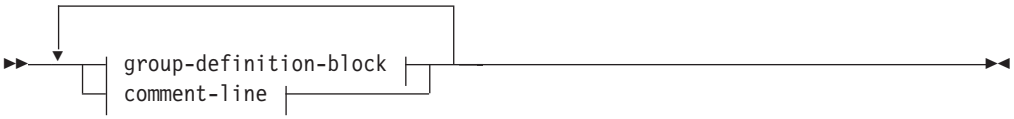
```
imothesn -f definition-file-name -ccsid ccsid
```

Here is a list of the code pages supported by an Ngram thesaurus:

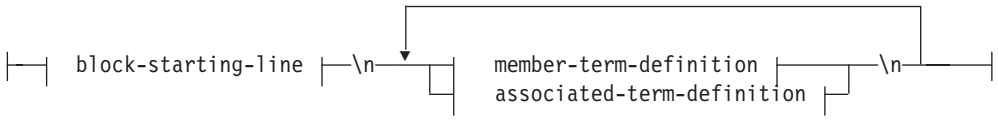
ASCII	850	PC Data Latin-1
	932	Combined Japanese
	942	Combined Japanese
	943	Combined Japanese
	949	Combined Korean
	950	Combined Traditional Chinese
	970	Combined Korean
	1363	Combined Korean
	1381	Combined Simplified Chinese
EBCDIC	00500	Latin-1
	00933	Korean
	00937	Traditional Chinese
	01388	Simplified Chinese
	05026	Japanese Katakana
	05035	Japanese Latin

`imothesn` produces thesaurus files having the same name as *definition-file-name* with the extensions mentioned above. The files are created in the same directory as the definition file. If there already exists a thesaurus with the same name in this directory it is overwritten without warning.

Specify the content of the thesaurus using the following syntax diagram:



group-definition-block:

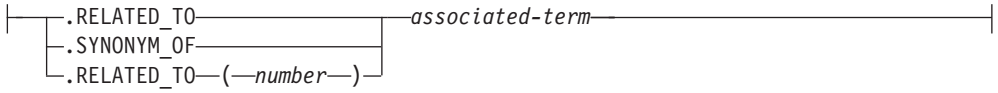


block-starting-line:

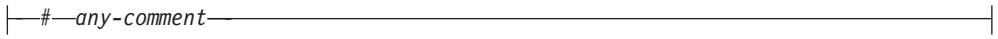
member-relation:

member-term-definition:

associated-term-definition:



comment-line:



Each member term must be written to a single line. Each associated term must be preceded by the relation name. The associated term is related to each member term with respect to the specified relation. If all member terms are related to each other, this can be specified using a member relation. Example:

```
:WORDS:SYNONYM
   reject
   decline
    RELATED_TO(1) accept
```

The length of member terms and associated terms is restricted to 164 characters. Single-byte characters and double-byte characters of the same letter are regarded as the same. Capital and small letters are not distinct. A term can contain a blank character but either the single-byte character period ″.″ or colon ″:″ can be used.

The user-defined relations are all based on the *associative* type. They are identified by unique numbers between 1 and 128. For a definition of the associative relation type, refer to “Ngram thesaurus relations” on page 13.

If an application wants to use symbolic names for their thesaurus relations instead of the relation name and number, it must administrate the mapping itself. For example, if the relation OPPOSITE_OF was defined as RELATED_TO(1), the application has to map this name to the internal relation name RELATED_TO(1). Refer to “EhwSearch” on page 125 to find out how to use thesaurus expansion in a query.

Chapter 3. Using the API functions

This chapter describes how the API functions can be used together in a logical sequence. A more detailed description of each function is given in alphabetical order in “Chapter 5. Calling the API functions” on page 37. These are the subjects described in this chapter:

- Listing the servers and starting a session
- Creating and setting up an index
- Working with an index
- Searching for documents
- Working with the result of a search
- Working with result views

Listing the servers and starting a session

Use **EhwListServers** to obtain a list of the available Text Search Engine search service names.

To use Text Search Engine information-retrieval services, run **EhwStartSession** to start a session with a particular Text Search Engine server.

When you no longer need a particular Text Search Engine server session, use **EhwEndSession** to close it.

The Text Search Engine server must be started before you can start a session.

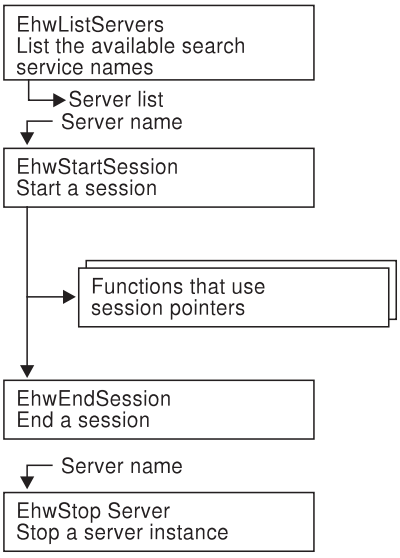


Figure 3. Listing the servers and starting a session

Creating and setting up an index

Many of the API functions work with an index and require the handle of an index as an input parameter. This section describes how to create and open an index.

Creating and deleting an index

This step describes how to create an empty index. Inserting document terms into the index is described in “Scheduling and indexing documents” on page 24.

Before you create the empty index, collect the following information:

- The index name
- The index type
- The name of the executable files for the library services server and client
- The location where you want the index to be stored
- The location where you want work files to be stored
- The Text Search Engine library server names of your installation.

Open a Text Search Engine session using **EhwStartSession**, then use **EhwCreateIndex** to create the index.

There is a related function, **EhwDeleteIndex**, that you would not normally use to delete a working index, but that can be useful for deleting test indexes.

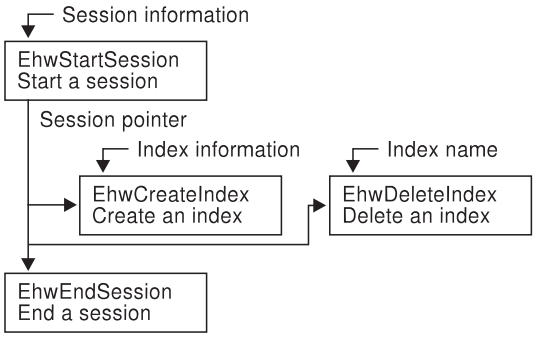


Figure 4. Creating and deleting an index

Getting the names of the existing indexes

Before you can work with an index you must open it. To do this, you must know the name of the index. Use **EhwListIndexes** to get the names of the indexes that are accessible through the Text Search Engine server with which you have established a session. If the list is long, a continuation condition indicates whether you need to repeat the function to get the entire list. The function **EhwCancelContinuation** is useful if you have reached a continuation condition, but nevertheless want to restart **EhwListIndexes** to get the list again from the beginning.

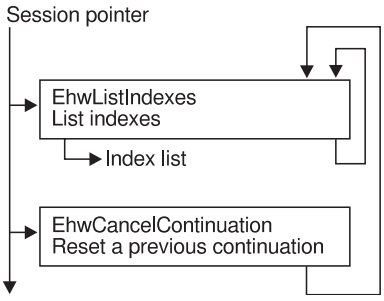


Figure 5. Getting the names of the existing indexes

Suspending and resuming an index

EhwSuspendIndex stops all administration tasks on and index, and prevent any further administration tasks on that index. This function can also be used to stop and prevent searches.

To allow the index to be used again, use the **EhwResumeIndex** function.

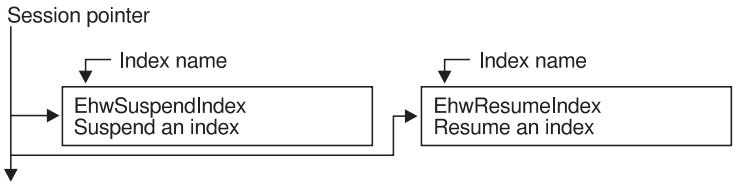


Figure 6. Suspending and resuming an index

Opening an index

You must open an index before you can use or update the information that it contains. Use **EhwOpenIndex** to open the index. This function provides an index handle that is used by other functions that process the index. These are described in “Working with an index” on page 24 and “Searching for documents” on page 26.

When you have finished working with an index, close it using **EhwCloseIndex**.

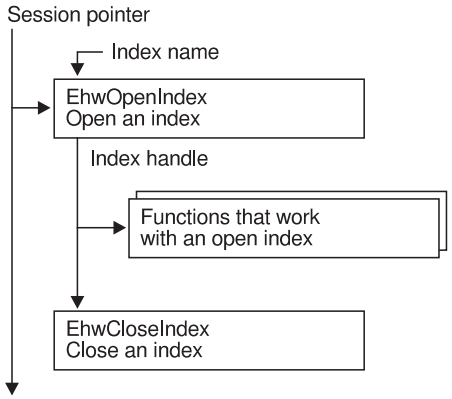


Figure 7. Opening an index

Getting index information

While an index is open, you can use **EhwGetIndexInfo** to verify the information that you specified when you created the index, such as the index name, type, and the names of the library services executables.

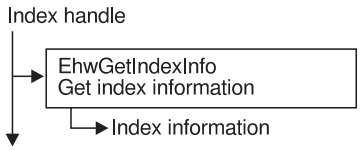


Figure 8. Getting index information

Setting and getting the default indexing rules

When indexing a document, Text Search Engine needs to know:

- Which type of document it is; for example a WordPerfect document, or a flat ASCII file
- Which language is the document written in
- Which Coded Character Set Identifier (CCSID) is used.

Text Search Engine can detect most of the supported document types automatically. If for some reason automatic recognition is not possible, Text Search Engine refers to default values that you establish.

Use the **EhwSetIndexingRules** function to define the default indexing rules.

There is a corresponding function **EhwGetIndexingRules** that lets you verify the rules settings that you have made.

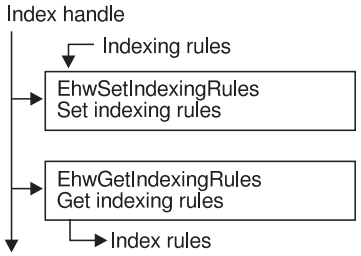


Figure 9. Setting and getting the default indexing rules

Working with an index

This section describes the following functions:

- Scheduling and indexing documents
- Getting the index status
- Improving performance and the use of storage
- Setting and getting the index function status.

Scheduling and indexing documents

Indexing is a two-step process:

1. Schedule the documents that are to be indexed, that is, put them in a queue for indexing. Use **EhwScheduleDocument** to do this.
2. Later, preferably when the system is lightly loaded, index the documents. Use **EhwUpdateIndex** to do this. This function starts a server task that updates the Text Search Engine index by processing the scheduled documents.

You can also schedule documents for deletion from an index.

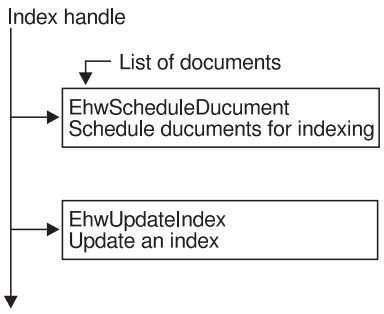


Figure 10. Scheduling and indexing documents

You can delete all scheduled entries from the indexing queue using **EhwClearScheduledDocuments**. To remove the complete contents of an index, use **EhwClearIndex**.

Getting the index status

Use **EhwGetIndexStatus** to get the following information about an index:

- The number of documents in the index
- The number of requests in the scheduling queue
- The number of messages that occurred during indexing.

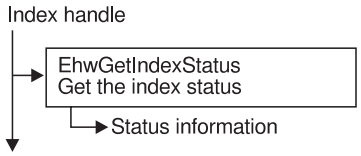


Figure 11. Getting the index status

Improving performance and the use of storage

Use the **EhwReorgIndex** function to start a server task that removes obsolete information from the index and to compress the information to improve performance and the use of storage.

Obsolete entries can occur in an index when documents that have previously been indexed are deleted. **EhwReorgIndex** checks for each term that its related document still exists. If the document no longer exists, the term is removed from the index. This function does not check whether documents exist in the library; it removes index entries that belong to documents whose identifier indicates that the document has been deleted.

When **EhwGetIndexStatus** returns the number of documents in an index, it distinguishes between a primary index and a secondary index. Text Search Engine looks in both parts of the index during a search. Each time you index documents, the indexed terms are stored in the secondary index. The size of the secondary index directly affects the indexing speed; as the secondary index grows, indexing becomes less efficient. So, from time to time, you should run **EhwReorgIndex** to move (merge) the indexed terms from the secondary index into the primary index.

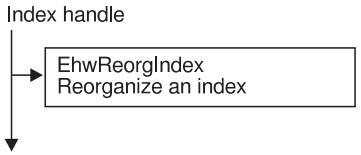


Figure 12. Improving performance and the use of storage

Setting and getting the index function status

EhwGetIndexFunctionStatus supplies status information about the Text Search Engine functions searching, queuing, indexing, and merging. The information indicates if the function is enabled, stopped, or running.

EhwSetIndexFunctionStatus allows you to change the status of these functions to enable or disable a function, or to reset a function in error.

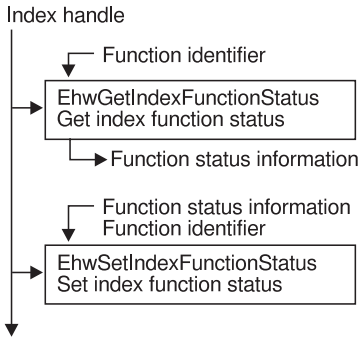


Figure 13. Setting and getting the index function status

Searching for documents

After you have opened a session and then opened one or more indexes, you can search for text. You can search in one index or in several. A search in more than one index concurrently is called a *cross-index search*.

You can restrict the scope of a search to certain documents or groups of documents.

The search functions

To search for text in a single index, in an open Text Search Engine session:
1. Open an index using **EhwOpenIndex**
2. Search for documents using **EhwSearch**
3. Close the index using **EhwCloseIndex**.

To make a cross-index search:
1. Open several indexes
2. Create an index group using **EhwCreateIndexGroup**
3. Search for documents
4. Delete the index group using **EhwDeleteIndexGroup**
5. Close the indexes.

You can specify a *query* to search the textual content of documents that are indexed in a particular Text Search Engine index. The result of a search, the search result, is a list of documents that match the query.

In the query, you can specify:
- One or several *search criteria* connected with Boolean operators
- Processing conditions, such as a time limit or a request for ranking information
- The scope of the query to restrict the result set.

The search criteria can be:
- A single *search argument* that must be found (or must not be found) in a qualifying document.
- Two or more search arguments that must satisfy a proximity condition, such as occurring within a single paragraph of a qualifying document.
- A free-text search argument that can consist of a single word, a phrase, or a sentence.

A search argument is one or several *search terms*, at least one of which must be found.

A search term is a word or phrase in a specified coded character set and language. You can mask a word or phrase in a search term by defining appropriate masking characters (also called global or wildcard characters).

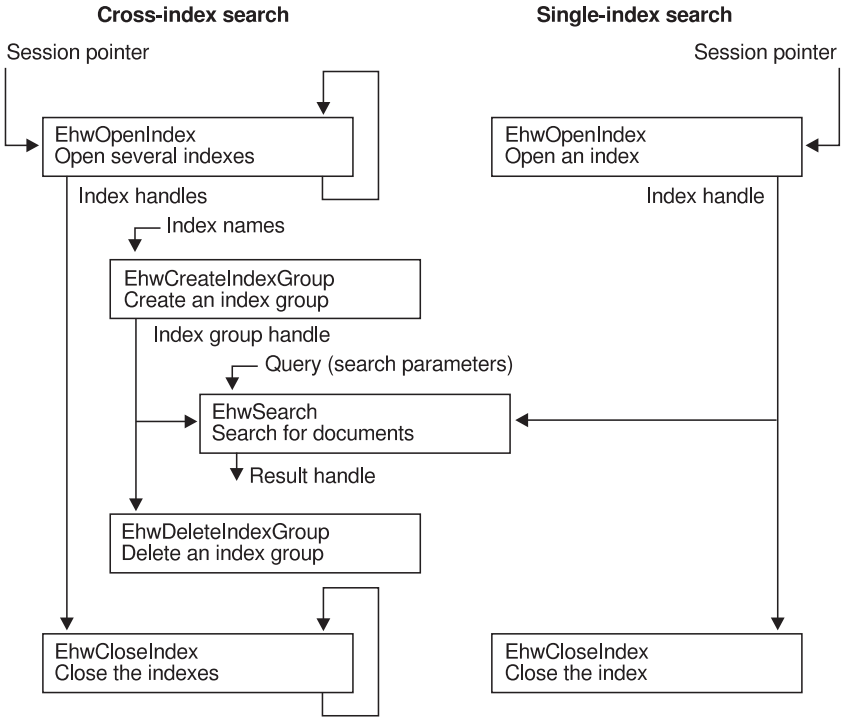


Figure 14. The search functions

Specifying the scope of a cross-index search

You can specify which documents or which groups of documents can be included in the result of a query. This is known as specifying the *scope* of a search. You specify a scope for a single-index query in the query itself. To specify a scope for a

cross-index search, however, you must use **EhwAddQueryScope**. The scope is valid only for the duration of the search.

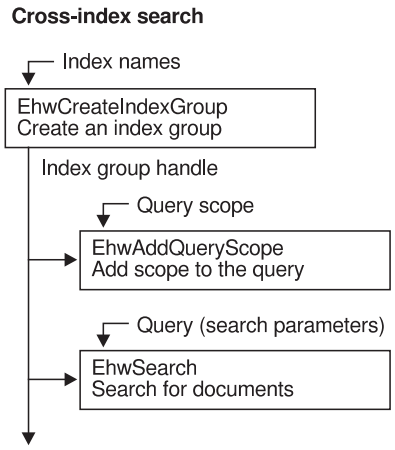


Figure 15. Specifying the scope of a cross-index search

Working with the result of a search

When a search is complete, a search result handle is available that lets you identify the documents that were found by the search.

When you no longer require search results, use **EhwDeleteResult** to release storage space.

Finding matches in search results

If you intend to develop a browser to view documents, it would be useful to highlight the found terms. This is an example for the use of **EhwGetMatches**. Use the function **EhwOpenDocument** and then use **EhwGetMatches** to get the text of the document and information for highlighting the search terms. The browse specifications determine which document is to be browsed. You can repeat this sequence to browse other documents.

EhwGetMatches gets portions of text. You can repeat it until the complete document has been brought in.

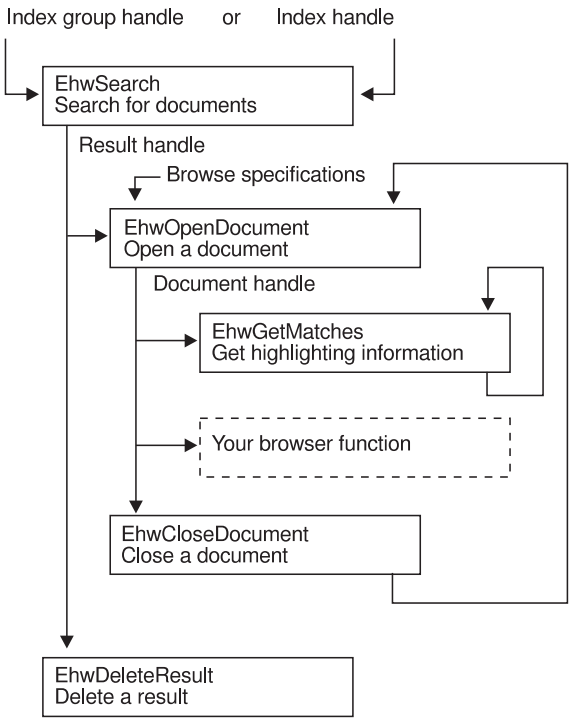


Figure 16. Browsing documents using your own browser

Getting problem information for a cross-index search

When you make a search on a group of indexes, the return code applies to the group. Use the function **EhwGetProblemInfo** to get the return codes for each index individually.

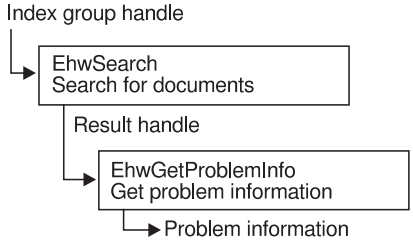


Figure 17. Getting problem information for a cross-index search

Working with result views

You can create a view of the found documents so that other API functions can select from this view, or sort the view, or cluster it, or use it to generate new query terms.

Creating and ranking a result view

Use **EhwCreateResultView** to create a view of the documents found by a search. Unlike **EhwListResult**, this function returns a handle that allows the result view to be used by other API functions.

Before you create the result view, you can use **EhwRank** to assign a rank value to each document. Note that a result can be ranked only if the `RANK` processing condition was used for the corresponding **EhwSearch** call, or if the query data stream included a freetext search argument. Whenever a result list is being ranked, the rank information is available in all views derived from this list. So **EhwRank** can be called both before and after a result view has been created.

After you have created the result view, run EhwSort to reorder the result by rank values.

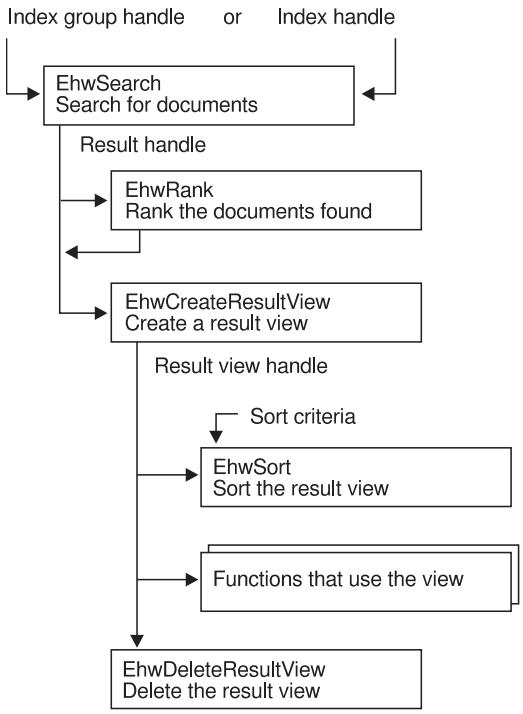


Figure 18. Creating and ranking a result view

Selecting from and sorting a result view

Use the function **EhwSelectResultView** to select some of the documents in the result view to create a subset in a new result view. Various selection criteria can be applied joined either with a Boolean AND or a NOT operator. Use **EhwSort** to sort a given result view according to various criteria, such as rack values. Note that **EhwSort**, unlike **EhwSelectResultView**, does not generate a new result view, but sorts an existing view.

The new view can be sorted and worked with in the same way as the original view created by **EhwCreateResultView**.

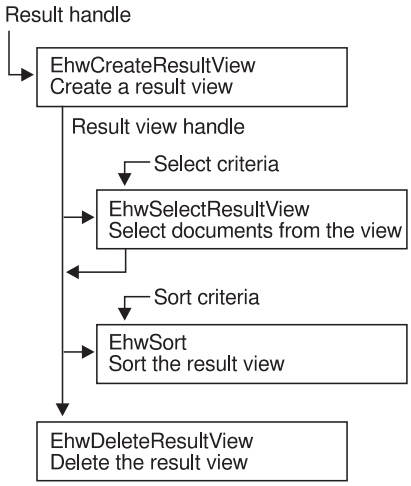


Figure 19. Selecting from and sorting a result view

Getting the contents of a result view

Use **EhwGetResultView** to get the contents of a result view created either by **EhwCreateResultView** or **EhwSelectResultView**. If the view is long, a continuation condition indicates whether you need to repeat the function to get the entire view. The function **EhwCancelContinuation** is useful if you have reached a continuation condition, but nevertheless want to restart **EhwGetResultView** to get the view again from the beginning.

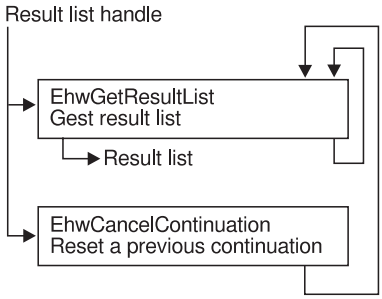


Figure 20. Getting the contents of a result view

Working with structured documents

Documents in a flat-file format may contain tags used to denote text sections, such as author or subject. Text Search Engine uses no predefined set of allowed tags. Instead, an application must define tags to be supported.

Not only flat files, but also HTML and XML document formats are supported. Elements in XML, and tags in HTML, correspond to section tags. For XML format, nesting of sections is supported.

For details, see the *Text Search Engine: Customization and Administration* manual.

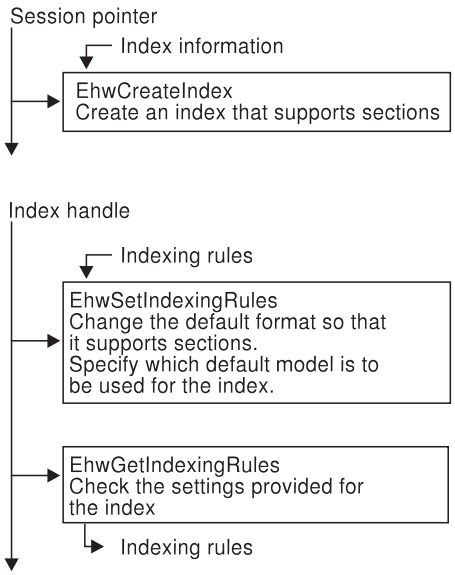


Figure 21. Creating an index enabled for section support

During EhwCreateIndex, the global model definition file is accessed.

An update on a section-enabled index is possible only after a call to EhwSetIndexingRules.

Chapter 4. Setting up your application

This chapter describes the environment where you can use the API, and explains how to set up your application for using Text Search Engine services. It also provides some general hints for programming the API function calls.

What the API provides

The Text Search Engine API can be used as an interface between application programs and Text Search Engine functions. It provides a general-purpose interface for obtaining the information retrieval services of Text Search Engine. The Text Search Engine server that provides these services can be installed on the local workstation, or on a remote workstation in a LAN environment (see “Environment and system overview” for more details).

The API works on a session basis. Before using any of the Text Search Engine information-retrieval requests, you must start a session with the Text Search Engine server. Outside a session, an application can list the Text Search Engine servers with which it can start a session from the particular client workstation.

Application programs using the Text Search Engine API must be written in the C programming language. Each Text Search Engine API function is invoked with a CALL statement (C function call). All arguments are passed to Text Search Engine separately as input parameters and results are returned as output parameters. Variable data, for both input and output, is passed in the form of data streams. For general conventions on parameter passing and for an explanation of Text Search Engine data streams, see “Data exchange conventions” on page 1.

Text Search Engine responds to each function call with a return code. In compliance with C programming conventions, the return code is the C function value. This indicates whether the call was successful, partially successful, or whether it caused an error condition. Where there are input errors, no corrective action is taken by Text Search Engine; instead, the call is rejected. Additional diagnosis information can also be returned, to indicate the cause of the error.

Some requests can produce large results. Where a result is too large to be handled conveniently in main storage, it is divided into blocks, and each block is passed separately to the application. In any case, Text Search Engine acquires the storage for the result of a function call and then passes the address of the storage to the application.

The processing of a request cannot be interrupted. However, a time limit or result limit can be specified for the processing of search requests.

Environment and system overview

Text Search Engine is designed for a LAN client-server environment and for stand-alone workstations. Depending on the special needs for work group management, a Text Search Engine installation in a LAN is flexible and can comprise:

- One or several Text Search Engine LAN servers
- Clients with access to one or several remote Text Search Engine servers
- Clients containing a local Text Search Engine server and having access to remote servers.

The API is available on any Text Search Engine client. Once the installation of any of the above combinations is set up, applications using the Text Search Engine API can obtain the services of any Text Search Engine server accessible from the local Text Search Engine client.

Note that the server must have been started before a connection is possible.

Figure 22 shows a typical configuration.

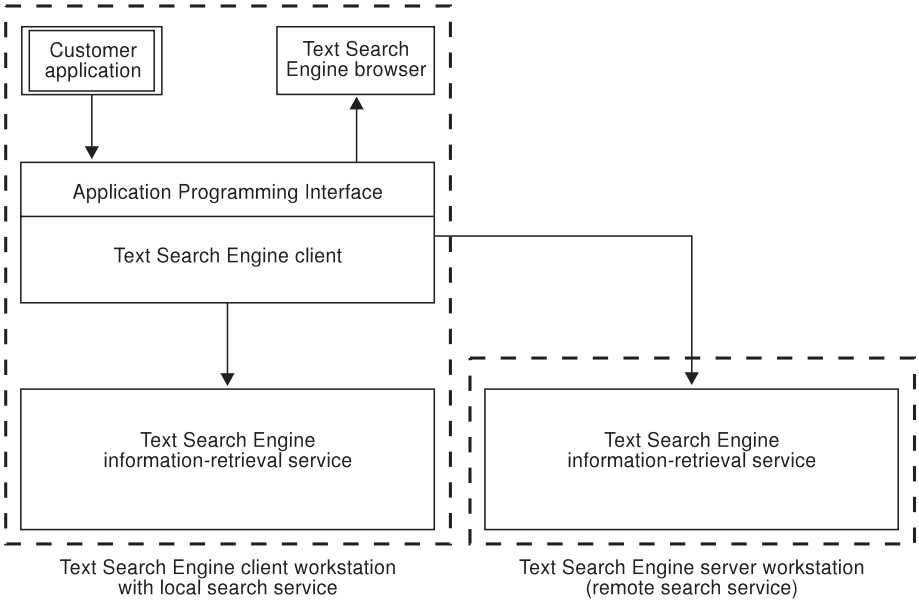


Figure 22. Text Search Engine sample configuration

Setting up your application

Do the following to use your application program with the Text Search Engine API:

1. Define symbolic names and prototypes
2. Link your application

Defining symbolic names

The symbolic name definitions and the function prototypes are provided in IMOAPIC.H as part of the API toolkit. The symbolic names are used for constants such as return codes, data stream item identifiers, and data stream item types. To include these definitions in your application program, add the following statement:

```
#include <imoapic.h>
```

Linking your application

To use your application program with the Text Search Engine API, link your program to the API.

For Windows NT systems

Text Search Engine API functions are contained in the dynamic link library IMOAPI.DLL. The external API function calls are resolved in the import library IMOAPI.LIB.

To link your application with the API dynamic link library IMOAPI.DLL use IMOAPI.LIB as the library file.

For UNIX systems

In AIX®, Text Search Engine API functions are contained in the shared library `libimoapi.a`.

In Solaris, Text Search Engine API functions are contained in the shared library `libimoapi.so`.

To link your application with this library, use the following link option with your linker:

```
-l imoapi
```

For z/OS systems

Text Search Engine API functions are reachable via the definition side deck `imoapi.x`.

To link your application, add `imoapi.x` to your linker input.

Some programming hints

Here are some points to consider when you are designing and writing the application program:

- All storage used for data stream output resulting from API function calls is reused or released by Text Search Engine with a subsequent call. Therefore, it is important to save any information that is required after a subsequent call.
- Input data streams should contain only items that are defined for the particular API function. Text Search Engine does not ignore unknown items. Any unknown items in an input data stream result in a syntax error.
- For output data streams, the application should ignore unknown items. This programming practice ensures that the data streams have no extensions.
- For reading and writing numeric fields in data streams, such as item lengths, item identifiers, and various item data fields, use the macros **VAL2()** and **ID()** in your application program. These C macros convert two-byte integer variables (VAL2) or constants (ID) from the big-endian format used in data streams to the local standard format and vice versa. These macros are provided with the Text Search Engine API toolkit in file IMOAPIC.H.

  These C macros also work in environments where the standard is not the little-endian format. So these macros should help you build portable applications.

Chapter 5. Calling the API functions

This chapter describes each function of the Text Search Engine API, giving the related input and output parameters, and explaining the parameters that must be provided or are returned in a data stream format. Each function description contains usage examples. The API functions are described in alphabetic order.

"Chapter 3. Using the API functions" on page 21 describes how the API functions can be used together in the sequence that they are intended to be used.

EhwAddQueryScope

This function adds information to a cross-index search that determines which documents or groups of documents, can be included in the result. The scope information is kept only until the cross-index search has been completed; the query scope has to be set again for each new EhwSearch call that searches on an index group.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

index group handle
The handle of the index group to be searched. This handle is returned by a previous EhwCreateIndexGroup call.

data stream length
The length, in bytes, of the data contained in the *query scope* parameter.

query scope
Query scope is not supported for libraries, it is supported only for file systems. The query scope information for filtering the search result. This parameter is supplied in a data stream format (see “Data stream syntax”).

Output parameters

diagnosis information
Returned only with return codes RC_DATASTREAM_SYNTAX_ERROR and RC_INDEX_NOT_MEMBER_OF_GROUP. It contains an offset value that points to a data stream item in the *query scope* parameter that caused the error. For example, the item may be out of sequence, or it may have an unknown identifier or type.

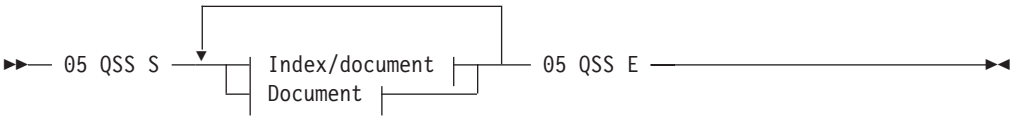
return code
The following return code values can be returned with this call:

RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_DATASTREAM_SYNTAX_ERROR
RC_QUERY_SCOPE_TOO_COMPLEX
RC_UNEXPECTED_ERROR
RC_NOT_ENOUGH_MEMORY
RC_INDEX_NOT_MEMBER_OF_GROUP

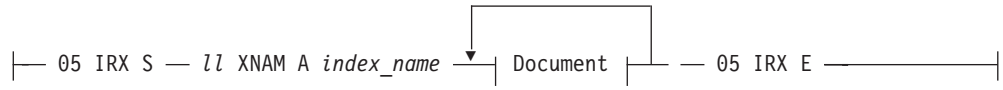
See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Query scope



Index/document:



Document:

The data stream items are:

QSS
Delimits the specification of a query scope. If you specify a query scope, the search result is limited to the set of documents defined by the scope. The query scope is either a list of document identifiers or a list of document group identifiers.

IRX
Delimits the scope information supplied for an index.

XNAM
The name of the index for which the scope is specified. Each index name must be a member of the input index group of the current session.

DID
Specifies the identifier of a document that is indexed in the information-retrieval index to be searched.

DGID
Specifies a document group identifier, such as a directory name in the platform's file system.

Note: The specification of document groups as a query scope assumes the naming convention where document identifiers (fully qualified file names) always start with the document group identifier (subdirectory name). You should not use this feature with library systems that have different naming conventions.

Usage

In a query, you specify criteria for searching the textual content of indexed documents. For a cross-index search the query scope has to be supplied separately using the EhwAddQueryScope API call. The scope setting is valid only for the duration of a search.

Example

This is an example of an EhwAddQueryScope function call:

```
/*-------------------------------------------------------------*/
/* Issue an API function call to add a scope to a query        */
/*-------------------------------------------------------------*/
ulReturnCode =
EhwAddQueryScope (pSession,         /* In  -- session pointer    */
          ulIndexGrpHandle,         /* In  -- index group handle */
          ulDataLength,             /* In  -- data stream length */
```

```
            pDataStream,            /* In  -- query scope for grp*/
            &ulDiagnosisInfo);      /* Out -- diagnosis info     */

   if (ulReturnCode != RC_DONE)     /* check the API return code */
   {
                                    /* handle the function error */
   }                                /* endif API call failed     */

   /*-----------------------------------------------------------------*/
```

If the query scope for one index of a group is desired, the area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would have a value of 47.

```
Symbolic notation              Hexadecimal representation

05 QSS   S                     0005 0069 E2
05 IRX   S                     0005 0032 E2
13 XNAM  A EHWINDEX            000D 003C C1 454857494E444558
14 DGID  A e:\group1           000E 006B C1 653A5C67726F757031
05 IRX   E                     0005 0032 C5
05 QSS   E                     0005 0069 C5
```

EhwCancelContinuation

This function resets the continuation mode of a preceding function call.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

return code
The following return code values can be returned with this call:

RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_REQUEST_IN_PROGRESS
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Usage

The EhwCancelContinuation call is required only to reset a pending service. It can be used after an EhwListIndexes or EhwListResult call with return code RC_CONTINUATION_MODE_ENTERED when you are not interested in receiving the remaining information.

Example

In the following example, the EhwCancelContinuation call ensures that a possible subsequent EhwListResult function call with the same result handle returns the first portion of the result list data stream.

```
/*-------------------------------------------------------------------*/
do
{
   ulReturnCode =
   EhwListResult (pSession,         /* In  -- session pointer      */
                  ulResultHandle,   /* In  -- result handle        */
                  &ulDataLength,    /* Out -- data stream length   */
                  &pDataStream,     /* Out -- document list        */
                  &ulDiagnosisInfo);/* Out -- diagnosis info.      */
                                    /* check the API return code   */
   if ((ulReturnCode != RC_DONE) &&;
       (ulReturnCode != RC_CONTINUATION_MODE_ENTERED))
   {
                                    /* handle the function error   */
   }                                /* endif API call failed       */

   /*-------------------------------------------------------------*/
   /* parse the result view data stream ...                       */
   /*-------------------------------------------------------------*/
```

```
                                    /* ...                          */
   if (Condition)                   /* check end of loop condition */
   {
      break;                        /* leave the loop prematurely  */
   }

}  while (ulReturnCode == RC_CONTINUATION_MODE_ENTERED)

if (ulReturnCode == RC_CONTINUATION_MODE_ENTERED)
{
   /*-----------------------------------------------------------------*/
   /* cancel the pending continuation mode                            */
   /*-----------------------------------------------------------------*/
   ulReturnCode = EhwCancelContinuation
                  (
                   pSession,        /* In  -- session pointer      */
                   pulDiagnosisInfo /* Out -- diagnosis info       */
                  );

   if (ulReturnCode != RC_DONE)     /* check the API return code   */
   {
                                    /* handle the function error   */
   }                                /* endif API call failed       */
}                                   /* endif continuation mode     */
/*-----------------------------------------------------------------*/
```

EhwClearIndex

This function removes all indexed terms from an information-retrieval index.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

index handle
Identifies the information-retrieval index to be cleared. It is the handle returned by EhwOpenIndex when the index is opened.

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

return code
The following return code values can be returned with this call:

RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_REQUEST_IN_PROGRESS
RC_INCORRECT_AUTHENTICATION (Text Search Engine for AIX only)
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_CONFLICTING_TASK_RUNNING
RC_NOT_ENOUGH_MEMORY
RC_NO_ACTION_TAKEN
RC_FUNCTION_IN_ERROR
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR
RC_LS_NOT_EXECUTABLE
RC_LS_FUNCTION_FAILED

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Usage

EhwClearIndex can be useful while you are trying out the system after installation or if you decide to change the index type. The only way to undo this action is to index all the documents again.

Note: While EhwClearIndex is running, the index is suspended and all other running functions (such as search and index documents) of this index are terminated. When EhwClearIndex has completed, the index is resumed.

If the index was suspended before the EhwClearIndex call, processing is terminated and the return code RC_INDEX_SUSPENDED is set.

Example

This is an example of an EhwClearIndex function call:

ClearIndex

```
/*-------------------------------------------------------------------*/
/* issue an API function call to clear an index                      */
/*-------------------------------------------------------------------*/
ulReturnCode =
EhwClearIndex (pSession,            /* In  -- session pointer        */
               ulIndexHandle,       /* In  -- index handle           */
               &ulDiagInfo);        /* Out -- diagnosis info         */

if (ulReturnCode != RC_DONE)        /* check the API return code     */
{
                                    /* distinguish the code types    */
   /*----------------------------------------------------------------*/
   /*                                  no error code:                */
   /* case RC_NO_ACTION_TAKEN:         index already empty           */
   /*----------------------------------------------------------------*/
   /*                                  function error:               */
   /* case RC_UNEXPECTED_ERROR:                                      */
   /* case RC_SERVER_BUSY:                                           */
   /* case RC_SERVER_CONNECTION_LOST:                                */
   /* case RC_COMMUNICATION_PROBLEM:                                 */
   /* case RC_NOT_ENOUGH_MEMORY:                                     */
   /* case RC_FUNCTION_IN_ERROR:                                     */
   /* case RC_IO_PROBLEM:                                            */
   /* case RC_WRITE_TO_DISK_ERROR:                                   */
   /*                                  handle the function error     */
   /*                                  ...                           */
   /*                                  stop processing / return      */
   /*----------------------------------------------------------------*/
   /* default:                         application error:            */
   /*                                  handle the internal error     */
   /*                                  ...                           */
   /*                                  stop processing / return      */
   /*----------------------------------------------------------------*/
}                                   /* endif API return code > 0     */
/*-------------------------------------------------------------------*/
```

EhwClearScheduledDocuments

This function empties the indexing queue of an information-retrieval index.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

index handle
Identifies the information-retrieval index that the queue to be cleared belongs to. It is the handle returned by EhwOpenIndex when the index is opened.

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

return code
The following return code values can be returned with this call:

RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_REQUEST_IN_PROGRESS
RC_INCORRECT_AUTHENTICATION (Text Search Engine for AIX only)
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_CONFLICTING_TASK_RUNNING
RC_NOT_ENOUGH_MEMORY
RC_NO_ACTION_TAKEN
RC_FUNCTION_IN_ERROR
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR
RC_LS_NOT_EXECUTABLE
RC_LS_FUNCTION_FAILED

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Usage

This function can be used to clear an indexing queue that has been filled with documents for testing, or to clear documents from an indexing queue that have unintentionally been placed there.

Example

This is an example of an EhwClearScheduledDocuments function call:

```
/*-------------------------------------------------------------*/
/* issue an API function call to clear the queue of an index   */
/*-------------------------------------------------------------*/
ulReturnCode =
EhwClearScheduledDocuments
              (pSession,           /* In  -- session pointer    */
               ulIndexHandle,      /* In  -- index handle       */
               &ulDiagInfo);       /* Out -- diagnosis info     */
```

```
if (ulReturnCode != RC_DONE)        /* check the API return code */
{
                                    /* distinguish the code types*/
 /*-------------------------------------------------------------*/
 /*                                  no error code:             */
 /*                                                             */
 /* case RC_NO_ACTION_TAKEN:         queue already empty        */
 /*                                  ...                        */
 /* case RC_CONFLICTING_TASK_RUNNING:                           */
 /*                                  active update | reorg. task */
 /*                                  ...                        */
 /*                                  continue with processing   */
 /*-------------------------------------------------------------*/
 /*                                  function error:            */
 /* case RC_UNEXPECTED_ERROR:                                   */
 /* case RC_SERVER_BUSY:                                        */
 /* case RC_SERVER_CONNECTION_LOST:                             */
 /* case RC_COMMUNICATION_PROBLEM:                              */
 /* case RC_NOT_ENOUGH_MEMORY:                                  */
 /* case RC_FUNCTION_IN_ERROR:                                  */
 /* case RC_IO_PROBLEM:                                         */
 /* case RC_WRITE_TO_DISK_ERROR:                                */
 /*                                  handle the function error  */
 /*                                  ...                        */
 /*                                  stop processing / return   */
 /*-------------------------------------------------------------*/
 /* default:                         application error:         */
 /*                                  handle the internal error  */
 /*                                  ...                        */
 /*                                  stop processing / return   */
 /*-------------------------------------------------------------*/
}                                   /* endif API return code > 0 */
/*-------------------------------------------------------------*/
```

EhwCloseDocument

This function closes a document opened by EhwOpenDocument, and releases the storage allocated for document text and highlighting information.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

document handle
Identifies the document handle. It is the handle returned by EhwOpenDocument.

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

return code
The following return code values can be returned with this call:

RC_DONE
RC_REQUEST_IN_PROGRESS
RC_LS_FUNCTION_FAILED
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_NOT_ENOUGH_MEMORY
RC_IO_PROBLEM
RC_UNEXPECTED_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Usage

When you call the EhwCloseDocument function, the document handle becomes obsolete, and all associated storage is released.

Example

This is an example of an EhwCloseDocument function call:

```
/*-------------------------------------------------------------------*/
/* close the document                                                */
/*-------------------------------------------------------------------*/
ulReturnCode =
EhwCloseDocument  (pSession,        /* In  -- session pointer      */
                   ulDocHandle,     /* In  -- document handle      */
                   pulDiagnosisInfo /* Out -- diagnosis info       */
                  );

if (ulReturnCode != RC_DONE)        /* check the API return code   */
{
                                    /* handle the function error   */
}                                   /* endif API call failed       */
/*-------------------------------------------------------------------*/
```

EhwCloseIndex

This function closes an information-retrieval index.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

index handle
Identifies the information-retrieval index to be closed. It is the handle returned by EhwOpenIndex when the index is opened. After closing the index, this handle and all associated result handles become obsolete.

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

return code
The following return code values can be returned with this call:

RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_MEMBER_OF_INDEX_GROUP
RC_REQUEST_IN_PROGRESS
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Usage

When you call the EhwCloseIndex function, all search results for this information-retrieval index or index group are deleted, all result handles become obsolete, and all associated storage is released.

Example

This is an example of an EhwCloseIndex function call:

```
/*-------------------------------------------------------------------*/
/* close the current information-retrieval index                     */
/*-------------------------------------------------------------------*/
ulReturnCode =
EhwCloseIndex (pSession,            /* In  -- session pointer      */
               ulIndexHandle,       /* In  -- index handle         */
               &ulDiagnosisInfo);   /* Out -- diagnosis info.      */

if (ulReturnCode != RC_DONE)        /* check the API return code   */
{
                                    /* handle the function error   */
}                                   /* endif API call failed       */
/*-------------------------------------------------------------------*/
```

EhwCreateDocumentModel

This function adds the definition of a new document model. The document model is either added to the server instance related to the current session, or is added to a section-enabled index opened previously and specified by means of an index handle.

If the value of the input parameter index handle is 0L, the document model definition is added to the document models available to the server instance. Subsequent calls to EhwCreateIndex can then make use of the new document model for section-enabled indexes. The name of the document model must be unique for the server instance.

If the value of the input parameter index handle is a valid handle returned from a previous call to EhwOpenIndex, the new model will be available only to the index specified by this handle. This index must have property XTPROP_SECTIONS_ENABLED. Future calls to EhwUpdate will take into account the new document model. The name of the document model must be unique for the index.

For Ngram indexes, the models used must not contain definitions of sections within sections (nested sections).

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

index handle
Specifies the information-retrieval index to use. It is the handle returned by EhwOpenIndex when the index is opened.

data stream length
The length, in bytes, of the data contained in the *document model definition* datastream.

document model definition
The definition of the document model to be added, either to the server instance or to a specific index. This parameter is supplied in datastream format. See “Data stream syntax”.

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

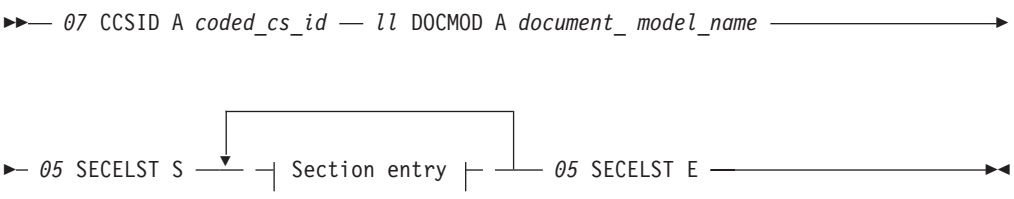
return code
The following return code values can be returned with this call:
RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_DATASTREAM_SYNTAX_ERROR
RC_INCORRECT_AUTHENTICATION (AIX only)
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST

RC_NOT_ENOUGH_MEMORY
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR
RC_CONFLICT_WITH_INDEX_TYPE
RC_DOCMOD_READ_PROBLEM
RC_DOCUMENT_MODEL_ALREADY_EXISTS
RC_SECTION_NAME_ALREADY_EXISTS
RC_SECTION_TAG_ALREADY_EXISTS
RC_INVALID_CHARACTER

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Document model definition



Section entry:

The data stream items are:

CCSID
Specifies the SAA Coded Character Set Identifier for the subsequent strings. File `IMOLANG.H` defines symbolic names for the CCSIDs. The 2-byte binary value must be specified in big-endian format.

DOCMOD
Specifies the name of the new document model. The name can contain the characters A-Z, a-z, 0-9.

SECELST
Delimits the list of all section entries defined for the new document model. Section names as well as section tags must be unique within the definition of a new model. Valid characters are A-Z, a-z, 0-9. For sections within sections, the character ’/’ is also allowed.

SECNAM
Name of a section to be defined for the new document model. This name can be used as input for EhwSearch calls.

SECTAG
String denoting the tag to be used for section recognition. During processing of EhwUpdate, the system looks for these tags in processed documents.

Usage

Use this call to add a new model definition either to an existing index or to the document models defined for the server instance.

Example

This is an example of an EhwCreateDocumentModel function call:

```
/*-----------------------------------------------------------------------------*/
/* Create document model for an information-retrieval index or a server instance */
/*-----------------------------------------------------------------------------*/
ulReturnCode =
EhwCreateDocumentModel(pSession,              /* In  -- session pointer      */
                       ulIndexHandle,         /* IN  -- index_handle         */
                       ulInDataLength,        /* In  -- data stream length   */
                       pInDataStream,         /* In  -- document model info  */
                       &ulDiagnosisInfo);     /* Out -- diagnosis info.      */

if (ulReturnCode != RC_DONE)                  /* check the API return code   */
{
                                              /* handle the function error   */
}                                             /* endif API call failed       */
/*-----------------------------------------------------------------------------*/
```

EhwCreateIndex

This function creates an index on the Text Search Engine server of the current session.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

data stream length
The length, in bytes, of the data contained in the *index information* parameter.

index information
Specifies the information-retrieval index creation information. This parameter is supplied in a data stream format (see “Data stream syntax” on page 53).

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

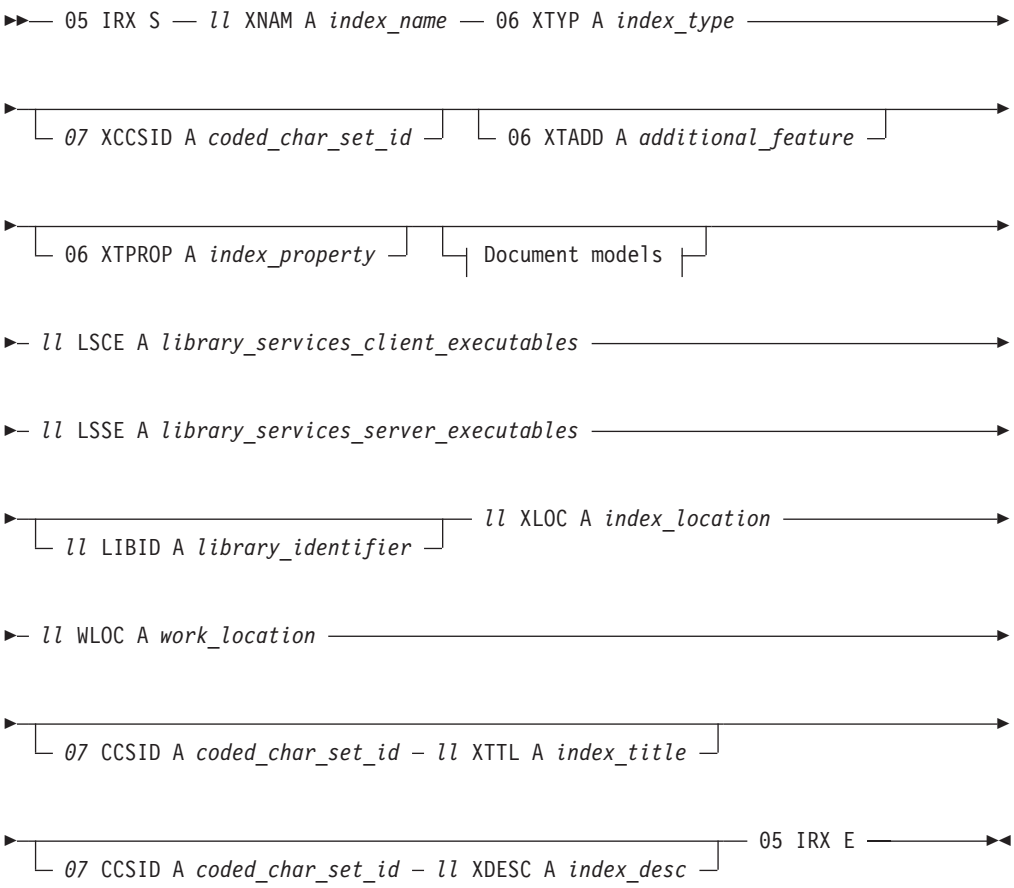
return code
The following return code values can be returned with this call:

RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_UNKNOWN_INDEX_TYPE
RC_INCORRECT_INDEX_NAME
RC_INCORRECT_LS_EXECUTABLES
RC_INCORRECT_LIBRARY_ID
RC_INCORRECT_LOCATION
RC_DATASTREAM_SYNTAX_ERROR
RC_REQUEST_IN_PROGRESS
RC_INCORRECT_AUTHENTICATION (Text Search Engine for AIX only)
RC_UNEXPECTED_ERROR
RC_INDEX_ALREADY_EXISTS
RC_MAX_NUMBER_OF_INDEXES
RC_LOCATION_IN_USE
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_NOT_ENOUGH_MEMORY
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR
RC_CCS_NOT_SUPPORTED
RC_CONFLICT_WITH_INDEX_TYPE
RC_DOCMOD_READ_PROBLEM
RC_UNKNOWN_DOCUMENT_MODEL_NAME

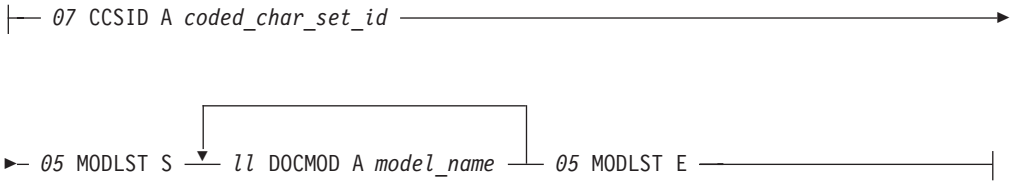
See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Index information



Document models:



The data stream items are:

IRX
Delimits the information supplied for the index.

XNAM
Specifies the name of the index to be created. The name can contain the characters A-Z, 0-9; lowercase characters are not allowed. Index names must be unique, and must not be longer than 8 characters.

XTYP
Specifies the type of this index. The *index_type* is a 1-byte code:

XTYP_LINGUISTIC
Specifies a linguistic index; that is, words are indexed in their base (lemma) forms; for example, `mouse` and `mice` are both indexed as `mouse`.

XTYP_PRECISE
Specifies a precise index; that is, words are indexed in the grammatical form in which they occur in the original text; for example, `mice` is indexed as `mice`.

XTYP_GTR

No other index type can support documents or queries in DBCS CCSIDs.

If DBCS documents contain SBCS text, the boundary between an SBCS and a DBCS word should contain a blank, not just a carriage return, to avoid unwanted concatenation of the two words.

Take, for example, a document that contains the following text:

```
Version
number
```

If either the last letter of `Version`, or the first letter of `number` were a DBCS character, then the text could be included in the index as the single word `Versionnumber`.

Now imagine a document containing the same text, but where there is either a blank character before the new line, or the new line begins with a blank character as in

```
Version
 number
```

Here, word separation is successful, and the index contains the two words `Version` and `number`.

XCCSID
Specifies the CCSID for an Ngram index.

Ngram indexes can be created using the following CCSIDs:

ASCII	00819	Latin-1
	00850	Latin-1
	00932	Combined Japanese
	00942	Combined Japanese
	00943	Combined Japanese
	00949	Combined Korean
	00950	Combined Traditional Chinese
	00970	Combined Korean
	01252	Latin-1
	01363	Combined Korean
	01381	Combined Simplified Chinese
	01383	Combined Simplified Chinese
	05039	Combined Japanese
EBCDIC	00500	Latin-1
	00933	Korean

	00937	Traditional Chinese
	01388	Simplified Chinese
	05026	Japanese Katakana
	05035	Japanese Latin

XTADD
Requests additional feature for the selected index type.

XTADD_CASE_ENABLED
Enables case-sensitive searches on this index, for all CCSID's supported by Ngram indexes. This is valid only for Ngram indexes.

XTADD_NORMALIZED
Valid only for index type XTYP_PRECISE. This index type is case insensitive. Exceptions to this are words in all uppercase, such as UK and US, which are indexed in uppercase.

XTPROP
Specifies that the index has a non-default property.

XTPROP_SECTIONS_ENABLED

Enables the index to keep information about the document structure, such as its title or author sections. Default for a section's type is full text.

Information on sections is extracted and stored at indexing time and can be used for later query restriction.

The description of EhwSearch function shows how to call for searches restricted to one or more sections. See "EhwSearch" on page 125.

Indexes created with this extension should use a default document format that enables structure recognition inside a document.

CCSID
Specifies the CCSID of the DOCMOD document model name(s). Valid only for indexes that are created using the `XTPROP_SECTIONS_ENABLED` option. For these indexes, a CCSID must be specified.

File `IMOLANG.H` defines symbolic names for the CCSIDs. The 2-byte binary value must be specified in big-endian format.

MODLST
Delimits a document model name or a list of document model names. Valid only for indexes that are created using the `XTPROP_SECTIONS_ENABLED` option. For these indexes, there must be at least one document model name, and its name must match a model that has been defined for the server instance. See "EhwCreateDocumentModel" on page 49.

The list must contain only the names of models defined for the server instance.

For Ngram indexes, the models must not contain sections within sections (nested sections).

DOCMOD
Document model name specifying which document model or models are to be used by the new index when adding documents to it.

Valid characters are A-Z, a-z, 0-9.

LSCE
Specifies the name of the client library services executable that is used with this index. The name can contain the characters A-Z, a-z, 0-9.

The client library service provided with Text Search Engine to support all file systems is called IMOLSCFS.

LSSE
Specifies the name of the server library services executable that must be used with this index. The name can contain the characters A-Z, a-z, 0-9.

The server library service to support the file system is called IMOLSSFS.

LIBID
Specifies the symbolic library identifier with which the information-retrieval index is associated.

If provided, this item has to be specified as an input parameter when the library services function LIB_init is called.

XLOC
Specifies the location of the index data.

WLOC
Specifies the location of the work data.

XTTL
Specifies the title of the information-retrieval index.

XDESC
Specifies the description of the information-retrieval index.

CCSID
Specifies the CCSID of the title and title description. A CCSID is mandatory if a title or title description exists. A list of valid CCSIDs is contained in `IMOLANG.H`.

Usage

Index names must be unique on a given server.

The index location and the work location specified with the EhwCreateIndex function call are reserved for this information-retrieval index only.

Note that directories for work location and index location must be unique for one index instance.

An Ngram index is created for use with one CCSID only. Requests like search or update fail if they use a CCSID different to the one it was created for.

Example

This is an example of an EhwCreateIndex function call:

```
/*-------------------------------------------------------------------*/
/* create a information-retrieval index                              */
/*-------------------------------------------------------------------*/
ulReturnCode =
EhwCreateIndex (pSession,           /* In  -- session pointer      */
                ulDataLength,       /* In  -- data stream length   */
                pDataStream,        /* In  -- index information    */
                &ulDiagnosisInfo);  /* Out -- diagnosis info.      */

if (ulReturnCode != RC_DONE)        /* check the API return code   */
{
```

```
                                    /* handle the function error   */
  }                                 /* endif API call failed       */
  /*-------------------------------------------------------------------*/
```

The area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would have a value of 108.

```
Symbolic notation              Hexadecimal representation

05 IRX   S                     0005 0032 E2
12 XNAM  A INDEX07             000C 003C C1 494E4445583037
06 XTYP  A XTYP_PRECISE        0006 003E C1 03
13 LSCE  A IMOLSCFS            000D 003F C1 4548534C53434653
13 LSSE  A IMOLSSFS            000D 0040 C1 4548534C53534653
27 XLOC  A \\SHL1293\          001B 0043 C1 5C5C53484C...
           INDEX07\DATA
27 WLOC  A \\SHL1293\          001B 0044 C1 5C5C53484C...
           INDEX07\WORK
05 IRX   E                     0005 0032 C5
```

EhwCreateIndexGroup

This function defines a group of indexes temporarily belonging together for the duration of a session. On this group of indexes, a cross-index search can be performed by using EhwSearch. The indexes must all be located on the same Text Search Engine server.

The types of the indexes in the group must be compatible; that is, the indexes must be one of the following:
- All precise and same extension
- All linguistic
- All Ngram with the same CCSID

If some of the indexes within a group are enhanced with additional features, then the query for a cross-index search must use the minimum subset of query keywords compatible with the index having the least features. For example, an index group containing indexes of type XTYP_LINGUISTIC and XTYP_LINGUISTIC with an enhancement via the XTPROP_SECTIONS_ENABLED feature are treated as if all indexes were of type XTYP_LINGUISTIC without any enhancements.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

data stream length
The length, in bytes, of the data contained in the *index names* parameter.

index names
Specifies the names of information-retrieval indexes already opened by an EhwOpenIndex call, to be grouped for the purpose of a cross-index search. The index names are supplied in a data stream format (see “Data stream syntax” on page 59).

At least two index names must be supplied.

Output parameters

index group handle
Identifies the information-retrieval index group and its status. The index group handle is used as input parameter for a search across multiple indexes.

diagnosis information
For the following return codes, the diagnosis information returned for this call contains the offset within the input data stream to the first index name in error.
RC_DATASTREAM_SYNTAX_ERROR
RC_CONFLICT_WITH_INDEX_TYPE

return code
The following return code values can be returned with this call:
RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_DATASTREAM_SYNTAX_ERROR
RC_NOT_ENOUGH_MEMORY
RC_UNEXPECTED_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Index names

The data stream item is:

XNAM
Specifies the name of an index that is to become a member of the group.

Usage

This function creates an index group which, after successful completion of the call, is represented by the index group handle. This handle can then be used to start a search across all the indexes belonging to the group. After deletion of an index group, all result handles and result list handles become obsolete. An index belonging to a group cannot be closed while its group exists.

Example

This is an example of an EhwCreateIndexGroup function call:

```
/*-------------------------------------------------------------*/
/* call API function EhwCreateIndexGroup                       */
/*-------------------------------------------------------------*/
   ulReturnCode =
   EhwCreateIndexGroup
                (pSession,         /* In  -- session pointer   */
                 &ulDataLength,    /* In  -- data stream length*/
                 &pDataStream,     /* In  -- index names list  */
                 ulIxGrpHandle,    /* Out -- group handle      */
                 &ulDiagnosisInfo);/* Out -- diagnosis info.   */
                                   /* check the API return code*/
   if (ulReturnCode != RC_DONE)
   {
                                /* handle the function error   */
   }                            /* endif API call failed       */

   /*-----------------------------------------------------------*/
   /* get the returned index group handle                       */
   /*-----------------------------------------------------------*/
```

The area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would have a value of 26.

```
Symbolic notation             Hexadecimal representation

13 XNAM A EHWINDEX            000D 003C C1 454857494E444558
13 XNAM A EHWINDX2            000D 003C C1 454857494E445832
```

EhwCreateResultView

This function creates a list of descriptive information for all documents of a specified search result list.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

result handle
Identifies the search result to be accessed. It is the handle returned by an EhwSearch call.

Output parameters

result list handle
A unique handle for the result list that includes the documents of the input result object.

result list size
The size of the result list.

diagnosis information
There is no diagnosis information returned for this call.

return code
The following return code values can be returned with this call:

RC_DONE
RC_EMPTY_LIST
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_NOT_ENOUGH_MEMORY
RC_RESULT_VIEW_EXISTS

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Usage

The result list is used by the functions EhwSelectResultView, EhwSort, EhwGetResultView, and EhwDeleteResultView.

An EhwCreateResultView call is made at most once for each result object. It creates the result list including all documents and their related information. When the result object is enhanced, for example through an EhwRank call, then the enhancement is valid for the result list created through EhwCreateResultView as well, regardless of the order of execution.

Example

This is an example of an EhwCreateResultView function call:

```
/*--------------------------------------------------------------------*/
/* call API function EhwCreateResultView                              */
/*--------------------------------------------------------------------*/
```

```
do
{
   ulReturnCode =
   EhwCreateResultView (pSession,       /* In  -- session pointer     */
                  ulResultHandle,       /* In  -- result handle       */
                  &ulResViewHandle,     /* Out -- result view handle  */
                  &ulResViewSize,       /* Out -- size of result view */
                  &ulDiagnosisInfo);    /* Out -- diagnosis info.     */
                                        /* check the API return code  */
   if (ulReturnCode != RC_DONE)
   {
                                  /* handle the function error      */
   }                              /* endif API call failed          */

   /*--------------------------------------------------------------------*/
   /* parse the result view data stream ...                              */
   /*--------------------------------------------------------------------*/

}  while (ulReturnCode == RC_CONTINUATION_MODE_ENTERED)
/*-----------------------------------------------------------------------*/
```

EhwDeleteDocumentModel

This function removes the specified document model from the server instance. The deleted document model is no longer be available for calls to EhwCreateIndex. If the document model is used by an existing index, this index is not affected by the deletion. You cannot delete a model defined for an existing index.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

data stream length
The length, in bytes, of the data contained in the *document model definition* datastream.

document model name
The name of the document model to be deleted from the server instance. This parameter is supplied in datastream format. See “Data stream syntax”.

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

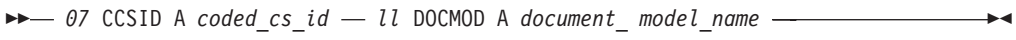
return code
The following return code values can be returned with this call:

RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_DATASTREAM_SYNTAX_ERROR
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_NOT_ENOUGH_MEMORY
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR
RC_UNKNOWN_DOCUMENT_MODEL_NAME

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Delete document model



The data stream items are:

CCSID
Specifies the SAA Coded Character Set Identifier for the subsequent strings. File `IMOLANG.H` defines symbolic names for the CCSIDs. The 2-byte binary value must be specified in big-endian format.

DOCMOD
Specifies the name of the document model to be removed from the server instance. The name can contain the characters A-Z, a-z, 0-9.

Usage

Use this call to delete a document model from the document models defined for the server instance defined by the *session pointer*.

Example

This is an example of an EhwDeleteDocumentModel function call:

```
/*--------------------------------------------------------------------------*/
/* Delete document model from a server instance                             */
/*--------------------------------------------------------------------------*/
ulReturnCode =
EhwDeleteDocumentModel    (pSession,       /* In  -- session pointer    */
                           ulDataLength,   /* In  -- data stream length */
                           pDataStream,    /* In  -- document model name*/
                           &ulDiagnosisInfo);  /* Out -- diagnosis info */

if (ulReturnCode != RC_DONE)  /* check the API return code              */
{                             /* handle the function error              */
}                             /* endif API call failed                  */
/*--------------------------------------------------------------------------*/
```

EhwDeleteIndex

This function deletes an index on the Text Search Engine server.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

data stream length
The length, in bytes, of the data contained in the *index name* parameter.

index name
Specifies the information-retrieval index to be deleted. This parameter is supplied in a data stream format (see “Data stream syntax”).

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

return code
The following return code values can be returned with this call:

RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_UNKNOWN_INDEX_NAME
RC_DATASTREAM_SYNTAX_ERROR
RC_REQUEST_IN_PROGRESS
RC_INCORRECT_AUTHENTICATION (Text Search Engine for AIX only)
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_SERVER_IN_ERROR
RC_INDEX_NOT_ACCESSIBLE
RC_NOT_ENOUGH_MEMORY
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Index name

The data stream item is:

XNAM
Specifies the name of the index to be deleted. It must be one of the names returned by the EhwListIndexes function for the current session.

Usage

EhwDeleteIndex not only clears the contents of an index, but also deletes the index itself. When you call the EhwDeleteIndex function, all data sets of the index location and the work location that were specified with EhwCreateIndex when you created the index, are deleted. If this index is open for the current session, it first has to be closed before it can be deleted. When the search service is integrated in a library, such as IBM Content Manager, you must first call EhwClearIndex to keep the library server information synchronized with the content of the document index.

Example

This is an example of an EhwDeleteIndex function call:

```
/*-------------------------------------------------------------------*/
/* call API function to delete the information-retrieval index     */
/*-------------------------------------------------------------------*/
ulReturnCode =
EhwDeleteIndex (pSession,           /* In  -- session pointer      */
                ulDataLength,       /* In  -- data stream length   */
                pDataStream,        /* In  -- index name           */
                &ulDiagnosisInfo);  /* Out -- diagnosis info.      */

if (ulReturnCode != RC_DONE)        /* check the API return code   */
{
                                    /* handle the function error   */
}                                   /* endif API call failed       */
/*-------------------------------------------------------------------*/
```

The area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would have a value of 13.

```
Symbolic notation              Hexadecimal representation

13 XNAM A EHWINDEX             000D 003C C1 454857494E444558
```

EhwDeleteIndexGroup

This function deletes the information-retrieval index group created by an EhwCreateIndexGroup call. When the group is deleted, all the data created after the generation of the index group handle, and associated with it, is also deleted.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

index group handle
Identifies the index group to be deleted. It is the handle returned by an EhwCreateIndexGroup call.

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

return code
The following return code values can be returned with this call:
RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_UNEXPECTED_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Example

This is an example of an EhwDeleteIndexGroup function call:

```
/*-------------------------------------------------------------------*/
/* call API function EhwDeleteIndexGroup                             */
/*-------------------------------------------------------------------*/
do
{
   ulReturnCode =
   EhwDeleteIndexGroup
                (pSession,         /* In  -- session pointer   */
                 ulIxGrpHandle,    /* In  -- group handle      */
                 &ulDiagnosisInfo);/* Out -- diagnosis info.   */
                                   /* check the API return code*/
   if (ulReturnCode != RC_DONE)
   {
                                /* handle the function error   */
   }                            /* endif API call failed       */
/*-------------------------------------------------------------------*/
/* This index group and its results are deleted now                  */
/*-------------------------------------------------------------------*/
```

EhwDeleteResult

This function releases all information associated with a specified search result, or deletes all results associated with an index or index group. However, if an index handle is an input parameter to EhwDeleteResult, and the associated index is also a member of one or more index groups, related search results belonging to these index groups are not deleted.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

result handle, index handle, or index group handle
Identifies the particular search result to be deleted, or the information-retrieval index whose results are all to be deleted.

- *result handle* is the handle returned by an EhwSearch call. Specify the result handle to delete a particular search result. After deleting a search result, its handle becomes obsolete. You can no longer call the EhwRank function for any document of this result. Also, working with the corresponding search result list is no longer possible, because any deletion of a search result implicitly deletes this search result list.
- *index handle* is the handle returned by EhwOpenIndex when the index is opened. Specify the index handle to delete all search results. After deleting all search results of a given index, all result handles associated with this index become obsolete.
- *index group handle* is the handle returned by EhwCreateIndexGroup when the group was created. Specify the index group handle to delete all group related search results. After deleting all search results of the group, all result handles associated with this index group become obsolete.

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

return code
The following return code values can be returned with this call:

RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_REQUEST_IN_PROGRESS
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Usage

Because each search result can occupy a large amount of storage space, you should delete any results that you no longer need.

Example

This is an example of an EhwDeleteResult function call:

```
/*-------------------------------------------------------------------*/
/* issue an API function call to delete a search result              */
/*-------------------------------------------------------------------*/
ulReturnCode =
EhwDeleteResult (pSession,          /* In  -- session pointer        */
                 ulResultHandle,    /* In  -- result handle          */
                 &ulDiagnosisInfo); /* Out -- diagnosis info         */

if (ulReturnCode != RC_DONE)        /* check the API return code     */
{
                                    /* handle the function error     */
}                                   /* endif API call failed         */
/*-------------------------------------------------------------------*/
```

EhwDeleteResultView

This function releases the information associated with the search result list specified by an input handle.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

result list handle
Identifies the particular search result list to be deleted. It is the handle returned by an EhwSelectResultView or an EhwCreateResultView call. When you delete a search result list, its handle becomes obsolete.

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

return code
The following return code values can be returned with this call:

RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_SERVER_CONNECTION_LOST
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Example

This is an example of an EhwDeleteResultView function call:

```
/*-------------------------------------------------------------------*/
/* issue an API function call to delete the current search result  */
/*-------------------------------------------------------------------*/
ulReturnCode =
EhwDeleteResultView (pSession,      /* In  -- session pointer      */
                     ulResViewHandle,   /* In  -- result view handle   */
                     &ulDiagnosisInfo); /* Out -- diagnosis info       */

if (ulReturnCode != RC_DONE)        /* check the API return code   */
{
                                    /* handle the function error   */
}                                   /* endif API call failed       */
/*-------------------------------------------------------------------*/
```

EhwDelIndexingMsgs

This function removes all indexing messages that were listed by an EhwGetIndexingMsgs call.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

index handle
Identifies the information-retrieval index for which indexing messages are to be removed. It is the handle returned by EhwOpenIndex when the index is opened.

Output parameters

return code
The following return code values can be returned with this call:

RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_REQUEST_IN_PROGRESS
RC_INCORRECT_AUTHENTICATION (Text Search Engine for AIX only)
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_NOT_ENOUGH_MEMORY
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Usage

Use the EhwDelIndexingMsgs call to remove all of the indexing messages.

Each time you retrieve messages, it is advisable to delete them to avoid the list of indexing messages becoming too large. This also helps you to keep track of the indexing messages that you actually retrieved.

Example

This is an example of an EhwDelIndexingMsgs function call:

```
/*-----------------------------------------------------------------*/
/* function call to delete   indexing messages                     */
/*-----------------------------------------------------------------*/
ulReturnCode =
EhwDelIndexingMsgs(pSession,     /* In  -- session pointer      */
                   ulIndexHdl,   /* In  -- index_handle         */
                   &ulDiagInfo); /* Out -- diagnosis info       */

if (ulReturnCode  != RC_DONE)    /* check the API return code   */
{
                                 /* handle the function error   */
}                                /* endif API call failed       */
```

EhwEndSession

This function ends the information-retrieval session established by calling EhwStartSession, and releases the storage allocated during the session.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session. After ending the session, this pointer and all associated handles become obsolete.

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

return code
The following return code values can be returned with this call:
RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_REQUEST_IN_PROGRESS
RC_UNEXPECTED_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Usage

EhwEndSession should always be called before ending the application, except when the previous EhwStartSession call failed.

Example

This is an example of an EhwEndSession function call:

```
/*--------------------------------------------------------------------*/
/* end the session                                                    */
/*--------------------------------------------------------------------*/
ulReturnCode =
EhwEndSession (pSession,            /* In  -- session pointer      */
               &ulDiagnosisInfo);   /* Out -- diagnosis info       */

if (ulReturnCode != RC_DONE)        /* check the API return code   */
{
                                    /* handle the function error   */
}                                   /* endif API call failed       */
/*--------------------------------------------------------------------*/
```

EhwGetDocumentModel

This function returns information on the definition of a document model.

The target of this call can be either the server instance related to the current session or a valid handle returned from any previous call to EhwOpenIndex on any section-enabled index.

If detailed information on the document model is to be retrieved for the server instance related to the current session, the value of the index handle must be 0L. If detailed information is to be retrieved for a section-enabled index, the index handle must be that of a previously opened index.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

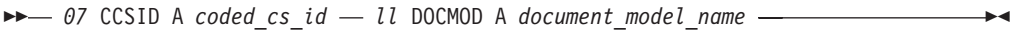
index handle
Specifies the information-retrieval index to use. It is the handle returned by EhwOpenIndex when the index is opened. If model information is requested from a server instance, this input parameter must be 0L.

data stream length
The length, in bytes, of the data contained in the *document model definition* datastream.

document model name
The name of the document model for which information is requested. This parameter is supplied in datastream format. See “Data stream syntax”.

Data stream syntax

Document model name



The data stream items are:

CCSID
Specifies the SAA Coded Character Set Identifier for the subsequent strings. File `IMOLANG.H` defines symbolic names for the CCSIDs. The 2-byte binary value must be specified in big-endian format.

DOCMOD
Specifies the name of the document model for which information is requested. The name can contain the characters A-Z, a-z, 0-9.

Output parameters

data stream length
The length, in bytes, of the data contained in the *document model information* datastream.

document model information
A list of all sections and their properties which are defined for the model

specified by the *document model name*. The output is returned in datastream format. See “Data stream syntax”.

diagnosis information
There is no diagnosis information returned for this call.

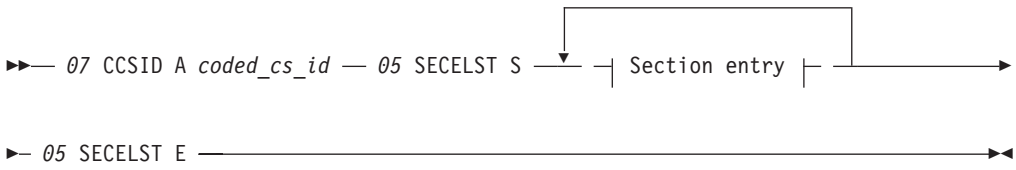
return code
The following return code values can be returned with this call:

RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_DATASTREAM_SYNTAX_ERROR
RC_INCORRECT_AUTHENTICATION (AIX only)
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_NOT_ENOUGH_MEMORY
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR
RC_CONFLICT_WITH_INDEX_TYPE
RC_DOCMOD_READ_PROBLEM
RC_CONTINUATION_MODE_ENTERED

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Document model information



Section entry:

The data stream items are:

CCSID
Specifies the SAA Coded Character Set Identifier for the subsequent strings. File `IMOLANG.H` defines symbolic names for the CCSIDs. The 2-byte binary value must be specified in big-endian format.

Depending on operating system, the values are EHWCCSID_00819 or EHWCCSID_00500.

SECELST
Delimits the list of all section entries defined for the document model. The list contains all sections of the document model specified by the document model name in the input datastream. The document model must have been defined for

the server instance or the index specified by the index handle (if the value of the index handle was given as 0L, or valid value as returned from EhwOpenIndex call respectively).

SECNAM
The name of a section as defined for the document model. This name can be used as input for EhwSearch calls.

SECTAG
A string denoting the tag to be used for section recognition. During processing of EhwUpdate, the parsers used for structured documents look for these tags.

Usage

Use this call to get information about the definition of a document model. If queries fail with RC_UNKNOWN_SECTION, use this call providing an index handle and document model name which were used for the query to get an overview of the model definition used for the index. For information on which default model is used for a specific index, see “EhwGetIndexingRules” on page 84.

Example

This is an example of an EhwGetDocumentModel function call:

```
/*-------------------------------------------------------------------------*/
/* Get information on document model used for an information-retrieval index */
/*-------------------------------------------------------------------------*/
ulReturnCode =
EhwGetDocumentModel   (pSession,              /* In  -- session pointer      */
                       ulIndexHandle,         /* IN  -- index_handle         */
                       ulInDataLength,        /* In  -- data stream length   */
                       pInDataStream,         /* In  -- index information    */
                       &ulOutDataLength,      /* Out -- data stream length   */
                       &pOutDataStream,       /* Out -- model information    */
                       &ulDiagnosisInfo);     /* Out -- diagnosis info.      */

if (ulReturnCode != RC_DONE)                  /* check the API return code   */
{
                                              /* handle the function error   */
}                                             /* endif API call failed       */
/*-------------------------------------------------------------------------*/
```

EhwGetIndexFunctionStatus

This function provides status information about the Text Search Engine functions search (EhwSearch) scheduling (EhwScheduleDocument), indexing (EhwUpdateIndex), and index merge (EhwReorgIndex). The output information indicates whether a function is enabled, stopped, or running. This information is available when an index is opened.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

index handle
Identifies the information-retrieval index for which function status information is to be provided. It is the handle returned by EhwOpenIndex when the index is opened.

function identifier
Identifies the Text Search Engine function for which status information is to be provided.

FCT_SEARCH_INDEX
Search function

FCT_SCHEDULE_DOCUMENTS
Schedule documents function

FCT_INDEX_DOCUMENTS
Index documents function

FCT_MERGE_INDEX
Reorganize index function.

Output parameters

data stream length
The length, in bytes, of the data contained in the *function information* parameter.

function information
Lists status information for the requested function. The information can contain time stamps of function events or error situations. This parameter is returned in a data stream format (see “Data stream syntax” on page 76).

diagnosis information
There is no diagnosis information returned for this call.

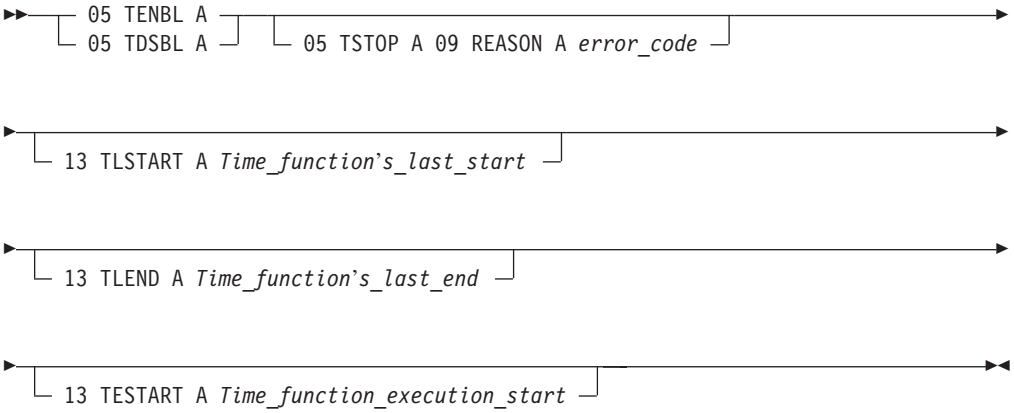
return code
The following return code values can be returned with this call:
RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_SERVER_CONNECTION_LOST
RC_REQUEST_IN_PROGRESS
RC_NOT_ENOUGH_MEMORY
RC_INDEX_SUSPENDED
RC_MAX_NUMBER_OF_INDEXES
RC_COMMUNICATION_PROBLEM
RC_UNEXPECTED_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

At least one of the items in the data stream must be present. “Usage” describes which items (for which function and for which function status) are contained in the data stream.

Function information



The data stream items are:

TENBL
Specifies the function enabled identifier

TDSBL
Specifies the function was disabled by an administrator

TSTOP
Specifies the function was stopped by the Text Search Engine server due to an internal error

REASON
Specifies the reason why a document could not be indexed, or why a problem occurred when a document was indexed. *error code* is a 4-byte unsigned long. For error codes and their description, see “Appendix B. Error codes returned by GetIndexingMsgs and GetIndexFunctionStatus” on page 207.

TLSTART
Specifies the time when the function was started last

TLEND
Specifies the time when the function ended last

TESTART
Specifies the time when the function started execution

Usage

Issue the EhwGetIndexFunctionStatus call to get the status of the following Text Search Engine functions:

Search (does not return TLSTART, TLEND, or TESTART)
Schedule documents

Index documents
Merge document index.

A data stream containing the status of these functions is returned. For the structure of the binary timestamp (8 bytes) returned by the function, see the file IMOLSDEF.H provided with Text Search Engine.

The items TLSTART, TLEND, and TESTART are not returned by the Search fucntion.

For the other functions, TLSTART and TLEND are returned only if the function for that index has already run, and TESTART is returned only when the function is actually running.

If TSTOP is returned, the function is faulty. The reason code can help to determine the error. Use EhwSetIndexFunctionStatus to reset the error condition.

Example

This is an example of an EhwGetIndexFunctionStatus call with function ID FCT_INDEX_DOCUMENTS.

```
/*-----------------------------------------------------------------*/
/* invoke API function to obtain the function status               */
/*-----------------------------------------------------------------*/
ulReturnCode = EhwGetIndexFunctionStatus
                 (pSession,       /* In  -- session pointer      */
                  ulIndexHdl,     /* In  -- index_handle         */
                  ulFunctionId,   /* In  -- function ID          */
                  &ulDataLength,  /* Out -- data stream length   */
                  &pStream,       /* Out -- admin. function info */
                  &ulDiagInfo);   /* Out -- diagnosis info       */

if (ulReturnCode  != RC_DONE)        /* check the API return code*/
{
                                     /* handle the function error*/
}                                    /* endif API call failed    */
/*-----------------------------------------------------------------*/
```

After completion of the function call, the area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would then have a value of 44.

```
Symbolic notation                Hexadecimal representation

05 TENBL   A                     0005 0051 C1
13 TLSTART A 1995101314200000    000D 0054 C1 07CB0A0D0E140000
13 TLEND   A 1995101314320000    000D 0055 C1 07CB0A0D0E200000
13 TESTART A 0000000000000000    000D 0056 C1 0000000000000000
```

EhwGetIndexInfo

This function provides information about an information-retrieval index. The index must have been opened before you can call EhwGetIndexInfo.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

index handle
Identifies the information-retrieval index. It is the handle returned by EhwOpenIndex when the index is opened.

Output parameters

data stream length
The length, in bytes, of the data contained in the *index detail information* parameter.

index detail information
Lists the index type, the client and server executables of library services, data path and work path of the index, and other index-specific information. This parameter is returned in a data stream format (see “Data stream syntax”).

diagnosis information
There is no diagnosis information returned for this call.

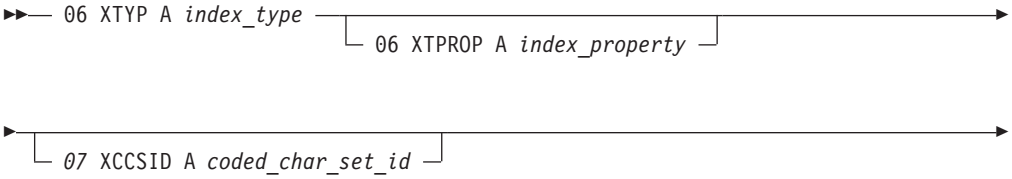
return code
The following return code values can be returned with this call:

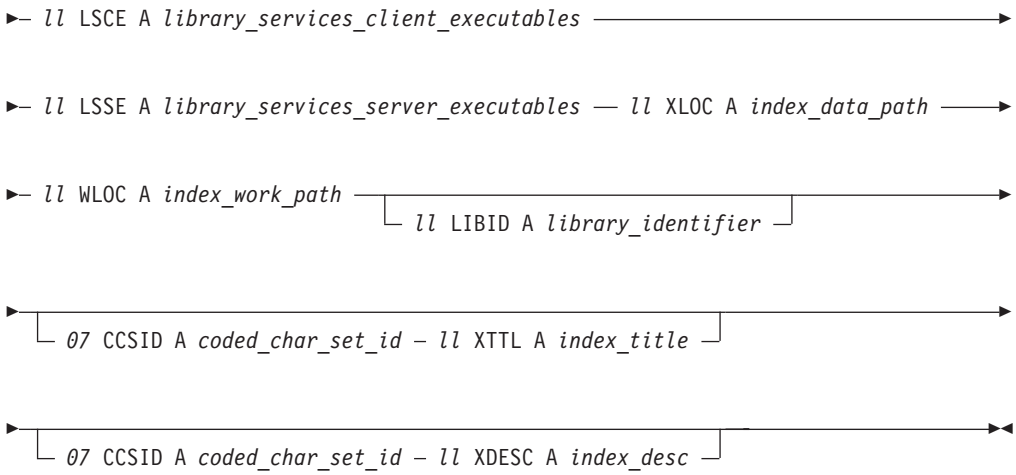
RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_NOT_ENOUGH_MEMORY
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Index detail information





The data stream items are:

XTYP
Specifies the type of this index. The *index type* is a 1-byte code:

XTYP_LINGUISTIC
Specifies a linguistic index; that is, words are indexed in their base (lemma) forms; for example, `mouse` and `mice` are both indexed as `mouse`.

XTYP_PRECISE
Specifies a precise index; that is, words are indexed in the grammatical form in which they occur in the original text; for example, `mice` is indexed as `mice`.

XTYP_GTR
Specifies an index not only for SBCS languages, but also for DBCS languages, known as an Ngram index.

XTPROP
Specifies that the index has a non-default property.

XTPROP_SECTIONS_ENABLED
An index with this property contains information about the document structure, such as title or author sections. For details on usage, see “EhwCreateIndex” on page 52.

XCCSID
Specifies the CCSID for an Ngram index.

LSCE
Specifies the name of the client library services executable that must be used with this index. The name does not include an extension.

The client library service provided with Text Search Engine to support all file systems is named IMOLSCFS.

LSSE
Specifies the name of the server library services executable that must be used with this index. The name does not include an extension.

The server library service is named IMOLSSFS.

XLOC
Specifies the data path where the index is located.

WLOC
Specifies the work path of the index.

LIBID
Specifies the symbolic library identifier with which the information-retrieval index is associated.

XTTL
Specifies the title of the information-retrieval index.

XDESC
Specifies the description of the information-retrieval index.

CCSID
Specifies the CCSID of the title and title description. A CCSID is mandatory if a title or title description exists.

Usage

Your application (or the users of your application) should know the index type to avoid specifying unsupported queries. Also, the default processing of queries varies for different index types. For details, refer to the EhwSearch function.

For information on the purpose and usage of library service functions, refer to "Chapter 6. Connecting Text Search Engine to a library" on page 159.

Example

This is an example of an EhwGetIndexInfo function call:

```
/*-------------------------------------------------------------------*/
/* invoke API function to obtain the index detail information      */
/*-------------------------------------------------------------------*/
ulReturnCode =
EhwGetIndexInfo (pSession,          /* In  -- session pointer      */
                 ulIndexHandle,     /* In  -- index handle         */
                 &ulDataLength,     /* Out -- data stream length   */
                 &pDataStream,      /* Out -- index list           */
                 &ulDiagInfo);      /* Out -- diagnosis info.      */

if (ulReturnCode != RC_DONE)        /* check the API return code   */
{
                                    /* handle the function error   */
}                                   /* endif API call failed       */
/*-------------------------------------------------------------------*/
```

After completion of the function call, the area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would then have a value of 58.

```
Symbolic notation             Hexadecimal representation

06 XTYP A XTYP_PRECISE        0006 003E C1 03
13 LSCE A IMOLSCFS            000D 003F C1 4548534C53434653
13 LSSE A IMOLSSFS            000D 0040 C1 4548534C53534653
13 XLOC A X:\PATH1            000D 0043 C1 583A5C5041544831
13 WLOC A X:\PATH2            000D 0044 C1 583A5C5041544832
```

EhwGetIndexingMsgs

This function lists indexing messages that can occur while Text Search Engine indexes documents.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

index handle
Identifies the information-retrieval index for which indexing messages are to be retrieved. It is the handle returned by EhwOpenIndex when the index is opened.

Output parameters

data stream length
The length, in bytes, of the data contained in the *indexing messages* parameter.

indexing messages
Lists the indexing messages of the specified information-retrieval index starting with the first message entry. This parameter is returned in a data stream format (see “Data stream syntax”).

diagnosis information
There is no diagnosis information returned for this call.

return code
The following return code values can be returned with this call:

RC_DONE
RC_MORE_INFORMATION
RC_EMPTY_LIST
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_REQUEST_IN_PROGRESS
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_NOT_ENOUGH_MEMORY
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR

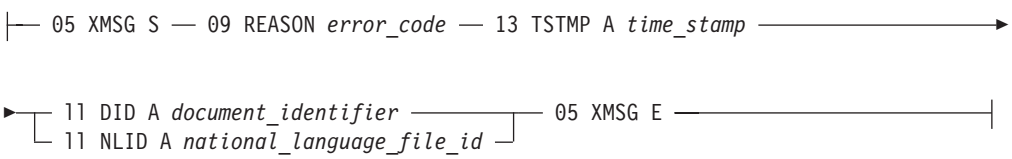
See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Indexing messages

GetIndexingMsgs

Indexing Message:



The data stream items are:

XMSG
Delimits one indexing message.

REASON
Specifies the reason why a document could not be indexed, or why a problem occurred when a document was indexed. *error code* is a 4-byte unsigned long. For error codes and their description, see “Appendix B. Error codes returned by GetIndexingMsgs and GetIndexFunctionStatus” on page 207.

TSTMP
Specifies the time information when the problem occurred. For a description of the format of *time stamp* see file IMOLSDEF.H provided with Text Search Engine.

DID
Specifies the identifier of the document causing the current indexing message.

NLID
Specifies the identifier of the file ID of the missing national language dictionary required to linguistically process one or more documents during one indexing run of Text Search Engine.

Usage

Indexing messages occur when, for example:

- Documents cannot be indexed
- Documents are indexed, but a problem occurs
- A language dictionary cannot be found.

If a query does not provide the expected result, for example, a result list does not contain documents that you know it should contain, use the EhwGetIndexingMsgs call to check if documents could not be indexed due to some problems. Also, as language dictionaries are needed for full linguistic processing, check if the dictionaries for the subject language are available for the indexing process.

After retrieving indexing messages, use the EhwDelIndexingMsgs call to delete some, or all, messages from the list that Text Search Engine maintains.

EhwGetIndexingMsgs tries to list all indexing messages available. When the return code RC_MORE_INFORMATION is returned, there are more messages to list. Before you can list these messages with another EhwGetIndexingMsgs call, you first have to delete all or some of the messages that you already have retrieved.

Example

This is an example of an EhwGetIndexingMsgs function call:

```
/*-------------------------------------------------------------*/
/* function call to retrieve indexing messages                 */
/*-------------------------------------------------------------*/
ulReturnCode =
EhwGetIndexingMsgs(pSession,     /* In  -- session pointer      */
                   ulIndexHdl,   /* In  -- index_handle         */
                   &ulLength,    /* Out -- length of data stream*/
                   &pchMsgTbl,   /* Out -- data stream          */
                   &ulDiagInfo); /* Out -- diagnosis info       */

if (ulReturnCode  != RC_DONE)    /* check the API return code   */
{
                                 /* handle the function error   */
}                                /* endif API call failed       */
```

EhwGetIndexingRules

This function lists the default indexing rules that Text Search Engine applies when format, language, and CCSID cannot be detected automatically for a document.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

index handle
Identifies the information-retrieval index for which the indexing rules apply. It is the handle returned by EhwOpenIndex when the index is opened.

Output parameters

data stream length
The length, in bytes, of the data contained in the *indexing rules* parameter.

indexing rules
Lists the indexing rules of the specified Text Search Engine index. This parameter is returned in a data stream format (see “Data stream syntax”).

diagnosis information
There is no diagnosis information returned for this call.

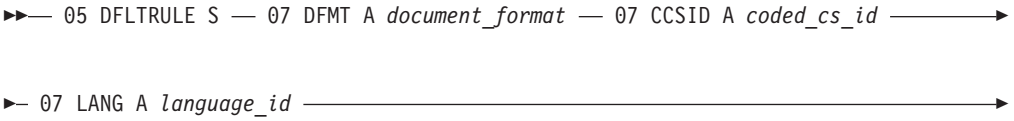
return code
The following return code values can be returned with this call:

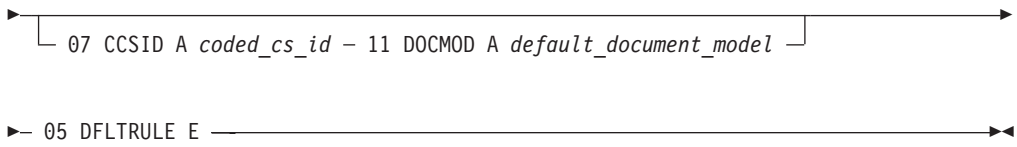
RC_DONE
RC_EMPTY_LIST
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_NOT_ENOUGH_MEMORY
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Indexing rules





The data stream items are:

DFLTRULE
Delimits the default indexing rule.

DFMT
Specifies the default document format that is assumed by Text Search Engine when the format cannot be determined automatically for a document. File IMOLSDEF.H, provided with Text Search Engine, defines symbolic names for all document formats that are supported by Text Search Engine. The 2-byte binary value is returned in big-endian format.

CCSID
Specifies the default SAA Coded Character Set Identifier that is applied by Text Search Engine when it cannot be determined automatically for a document. File IMOLANG.H, provided with Text Search Engine, defines symbolic names for all CCSIDs that are supported by Text Search Engine. The 2-byte binary value is returned in big-endian format.

LANG
Specifies the default language that is assumed by Text Search Engine when it cannot be determined automatically for a document. File IMOLANG.H, provided with Text Search Engine, defines symbolic names for all language identifiers that are supported by Text Search Engine. The 2-byte binary value is returned in big-endian format.

CCSID
Specifies the SAA Coded Character Set Identifier for the following string.

DOCMOD
Specifies the default model to be used for section-enabled indexes.

Usage

When indexing a document, Text Search Engine needs to know:

- Which type of document it is; if it is an AmiPro document, for example, or a flat ASCII file
- Which language the document is written in
- Which CCSID is used.

Most document formats are detected by Text Search Engine automatically, but a document does not always contain information about the language and CCSID that it is written in. So Text Search Engine needs a default indexing rule that is applied whenever the necessary information about the document format, language, and CCSID cannot be determined automatically.

Use the EhwGetIndexingRules call to check which indexing rules are currently active. For Ngram indexes, the value of CCSID is the one the index was created for. Documents written in a different CCSID cannot be indexed.

For indexes that have addditional type XTPROP_SECTIONS_ENABLED, addditional output consists of the name of the document model being used during

EhwUpdate processing. The CCSID preceding it gives the value of the SAA Coded Character Set Identifier used for encoding the string.

Check that the document format and document model agree. If the document format has been set correctly, but the chosen model defines a set of tags that are not valid for that document format, the index will not contain section information.

Example

This is an example of an EhwGetIndexingRules function call:

```
/*-------------------------------------------------------------*/
/*  Invoke API function to get the default indexing rules      */
/*-------------------------------------------------------------*/
ulReturnCode =
EhwGetIndexingRules(pSession,    /* In  -- session pointer      */
                    ulIndexHdl,  /* In  -- index_handle         */
                    &ulLength,   /* Out -- length of data stream*/
                    &pchRule,    /* Out -- data stream          */
                    &ulDiagInfo);/* Out -- diagnosis info       */

if (ulReturnCode  != RC_DONE)        /* check the API return code*/
{
                                     /* handle the function error*/
}                                    /* endif API call failed    */
```

EhwGetIndexStatus

This function returns a data stream containing information about the number of documents in an information-retrieval index, the number of requests in the indexing queue, the number of document messages written during indexing, and if the index is currently active. If the index is suspended, the return code indicates the time when an attempt was made to open or access the index.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

index handle
Identifies the information-retrieval index for which status information is to be provided. It is the handle returned by EhwOpenIndex when the index is opened.

Output parameters

data stream length
The length, in bytes, of the data contained in the *index status information* parameter.

index status information
Lists status information for the requested index. This parameter is returned in a data stream format (see “Data stream syntax”).

diagnosis information
There is no diagnosis information returned for this call.

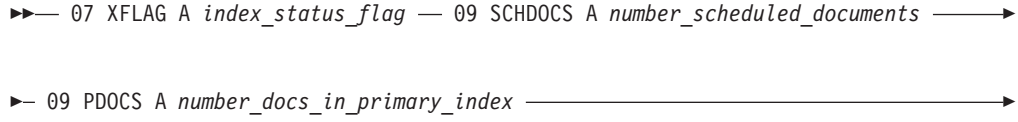
return code
The following return code values can be returned with this call:

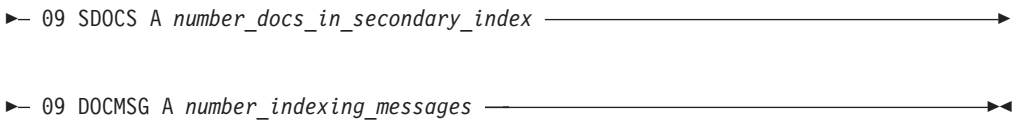
RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_SERVER_CONNECTION_LOST
RC_REQUEST_IN_PROGRESS
RC_NOT_ENOUGH_MEMORY
RC_INDEX_SUSPENDED
RC_MAX_NUMBER_OF_INDEXES
RC_COMMUNICATION_PROBLEM
RC_UNEXPECTED_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Index status information





The data stream items are:

XFLAG
Index status flag.

XFLAG_READ_WRITE
The index is active, all functions are available.

XFLAG_READ_ONLY
The index is suspended and is in a read-only mode, that is, EhwGetIndexFunctionStatus API calls are permitted. If the index is suspended totally, RC_INDEX_SUSPENDED is returned.

SCHDOCS
The number of indexing requests scheduled.

PDOCS
The number of documents in the primary index.

SDOCS
The number of documents in the secondary index.

If this number becomes large compared to the number of documents in the primary index, use EhwReorgIndex to merge the primary and secondary indexes to improve performance in searching for and indexing documents.

DOCMSG
The number of document messages written during indexing. Use EhwGetIndexingMsgs to get the content of these messages.

Usage

To get the status of an information-retrieval index, issue an EhwGetIndexStatus call. A data stream with the above described items is returned.

Note that, if an index is suspended totally, there are no data stream items returned but a return code indicating the suspension.

Example

This is an example of an EhwGetIndexStatus call. At the time of the call, the index is active and there are documents in the index. The indexing queue is not empty.

```
/*-----------------------------------------------------------------*/
/* Invoke API function to obtain the function status               */
/*-----------------------------------------------------------------*/
ulReturnCode =
EhwGetIndexStatus( pSession,      /* In  -- session pointer      */
                   ulIndexHdl,    /* In  -- index handle         */
                   &ulDataLength, /* Out -- data stream length   */
                   &pStream,      /* Out -- index status info    */
                   &ulDiagInfo);  /* Out -- diagnosis info       */

if (ulReturnCode  != RC_DONE)        /* check the API return code*/
{
```

```
                                    /* handle the function error*/
}                                   /* endif API call failed    */
/*-----------------------------------------------------------------*/
```

After completion of the function call, the area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would then have a value of 043.

```
Symbolic notation                Hexadecimal representation

07 XFLAG   A XFLAG_READ_WRITE    0007 0049 C1 0002
09 SCHDOCS A 25                  0009 004A C1 00000019
09 PDOCS   A 100                 0009 004B C1 00000064
09 SDOCS   A 5                   0009 004C C1 00000005
09 DOCMSG  A 3                   0009 004D C1 00000003
```

EhwGetMatches

This function returns the text of the document and highlighting information for all matches of the corresponding query.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

document handle
Identifies the document handle. It is the handle returned by EhwOpenDocument.

Output parameters

data stream length
The length, in bytes, of the data contained in the *document text* parameter.

document text
Contains the document text and the highlighting information for all matches. This parameter is returned in a data stream format (see “Data stream syntax”).

diagnosis information
There is no diagnosis information returned for this call.

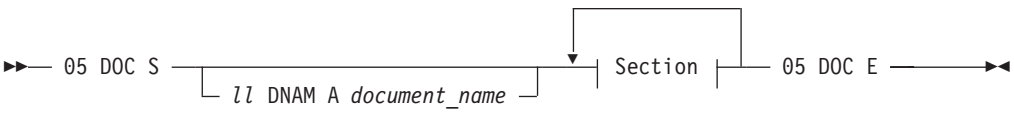
return code
The following return code values can be returned with this call:

RC_DONE
RC_CONTINUATION_MODE_ENTERED
RC_DICTIONARY_NOT_FOUND
RC_UNKNOWN_SESSION_POINTER
RC_REQUEST_IN_PROGRESS
RC_INCORRECT_HANDLE
RC_LS_FUNCTION_FAILED
RC_NOT_ENOUGH_MEMORY
RC_CAPACITY_LIMIT_EXCEEDED
RC_UNEXPECTED_ERROR
RC_IO_PROBLEM
RC_GTR_ERROR

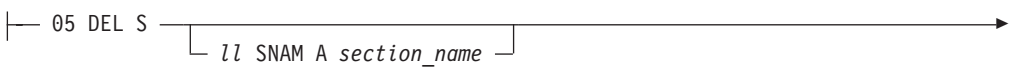
See “Appendix A. The API return codes” on page 191 for more information about these codes.

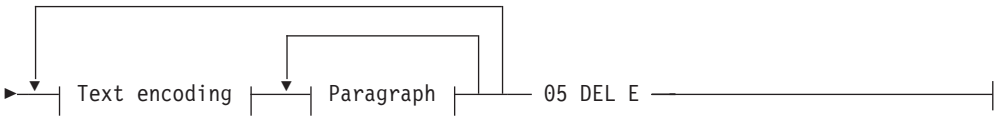
Data stream syntax

Document text

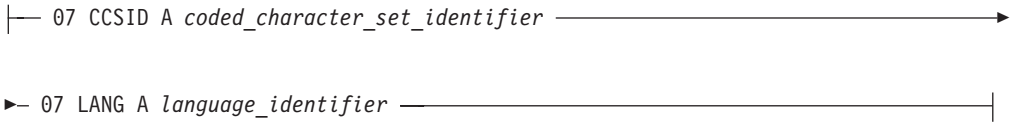


Section:



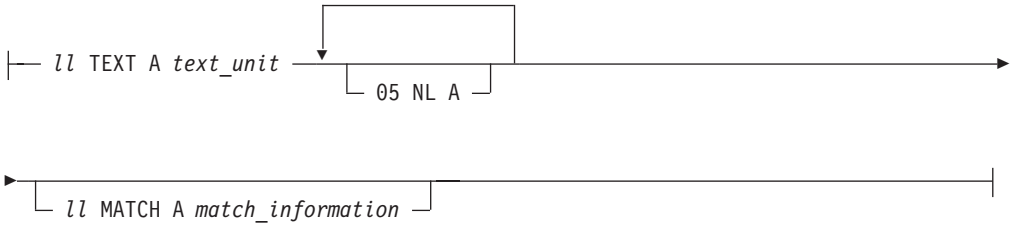


Text encoding:



Paragraph:

Paragraph text:



The data stream items are:

DOC
Indicates the start and end of a document.

DNAM
Specifies a document name. If no name is specified, the identifier of the document is used.

DEL
Indicates the start and end of a document element. The type of document element supported is a *text section*.

SNAM
Specifies the name of a text section.

PAR
Indicates the start and end of a text paragraph within the current section.

TEXT
Specifies one text portion within the current paragraph. Usually, *text unit* contains one line of text, and the TEXT item is followed by an NL item; but text

lines can also be split into several parts, each part specified in its own TEXT item. The text uses the CCSID and language associated with the current paragraph.

NL
Indicates the start of a new line in the current paragraph.

MATCH
Contains occurrence information for matches in the current text portion. The information is supplied as a sequence of binary number pairs. The first number in each pair is the offset of a match within the current text portion, the second number is the length (in bytes) of that match. The given length can exceed the length of the given text portion. Both offset and length are 2-byte values specified in big-endian format.

CCSID
Specifies the SAA Coded Character Set Identifier for text in the following paragraphs, which remains valid until a paragraph is preceded by a new CCSID item. The following CCSIDs are returned:

CCSID_00500
For text in the Latin-1 EBCDIC CCSID 500.

CCSID_0850
For text in the Latin-1 ASCII CCSID 850.

CCSID_00819
For text in the ASCII CCSID 819 as defined by ISO standard 8859-1.

CCSID_00942
For text in the Shift-JIS CCSID for combined Japanese.

CCSID_00949
For text in CCSID for combined Korean.

CCSID_00950
For text in CCSID for combined Traditional Chinese.

CCSID_01381
For text in CCSID for combined Simplified Chinese.

The symbolic names for CCSIDs are defined in the file IMOLANG.H of the library services toolkit provided with Text Search Engine. The 2-byte binary value is specified in big-endian format.

LANG
Specifies the language identifier for text in the subsequent paragraphs. It is valid until a paragraph is preceded by a new LANG item. File IMOLANG.H in the library services toolkit defines symbolic names for all language identifiers supported by Text Search Engine. The 2-byte binary value is specified in big-endian format.

The symbolic names for the item identifiers of the above data stream items are defined in the file IMOLSDEF.H of the library services toolkit provided with Text Search Engine.

Usage

This function allows you to get the text of a document and the highlighting information for use by your own browser program. EhwGetMatches returns the text in portions. When all portions of the document text have been returned, the return code RC_DONE indicates the end of the document.

Note: When RC_DONE is returned, and you want to repeat EhwGetMatches from the beginning of the document, you must first close the document and open it again. This is unlike other functions which automatically start from the beginning of the document the next time they are called after RC_DONE.

Example

This is an example of an EhwGetMatches function call:

```
/*-------------------------------------------------------------------*/
/* Get the document text with matches                                */
/*-------------------------------------------------------------------*/
ulReturnCode =
EhwGetMatches     (pSession,        /* In  -- session pointer      */
                   ulDocHandle,     /* In  -- document handle      */
                   pulDataLength,   /* Out -- data stream length   */
                   ppDataStream,    /* Out -- text and matches     */
                   pulDiagnosisInfo /* Out -- diagnosis info       */
                  );

if (ulReturnCode != RC_DONE)        /* check the API return code   */
{
                                    /* handle the function error   */
}                                   /* endif API call failed       */
/*-------------------------------------------------------------------*/
```

After completion of the last function call, the area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would have a value of 96.

```
Symbolic notation              Hexadecimal representation

05 DOC   S                     0005 019A E2
05 DEL   S                     0005 01A4 E2
07 CCSID A CCSID_0850          0007 0073 C1 1352
07 LANG  A LANG_ENU            0007 0074 C1 177B
05 PAR   S                     0005 01AE E2
16 TEXT  A First line.         0010 01B8 C1 466972...
05 NL    A                     0005 01BC C1
17 TEXT  A Match found.        0011 01B8 C1 4D6174...
05 NL    A                     0005 01BC C1
09 MATCH A 0005                0009 01C2 C1 00000005
05 PAR   E                     0005 01AE C5
05 DEL   E                     0005 01A4 C5
05 DOC   E                     0005 019A C5
```

EhwGetProblemInfo

When EhwSearch searches across several indexes and one or more indexes report problems, the EhwSearch return code applies to the index group as a whole. When this happens, you can use EhwGetProblemInfo to get specific error information about one information-retrieval index.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

result handle
Identifies the search result handle supplied by a previous EhwSearch call.

Output parameters

data stream length
The length, in bytes, of the data contained in *problem information*.

problem information
Lists the return code for each index that had problems during a cross-index search on an index group. This parameter is returned in a data stream format (see “Data stream syntax” on page 78).

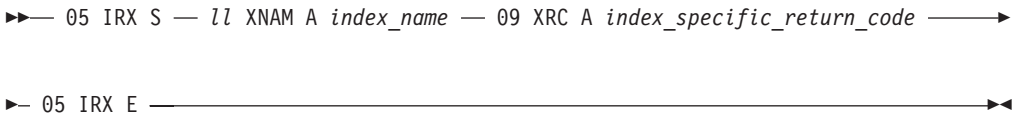
diagnosis information
There is no diagnosis information returned for this call.

return code
The following return code values can be returned with this call:
RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_CONFLICTING_TASK_RUNNING
RC_EMPTY_LIST
RC_UNEXPECTED_ERROR

See also “Appendix A. The API return codes” on page 191.

Data stream syntax

Problem information



The data stream items are:

IRX
Delimits the information supplied for the index.

XNAM
Name of the index for which an error occurred.

XRC
Specifies the index-specific return code.

Usage

When a cross-index search is started by EhwSearch on an index group, errors may occur. To determine the reason for each index-specific error, use EhwGetProblemInfo to find which index caused the problem. On return, the error return codes and the related index names are supplied in a data stream format.

Even if the result is empty because all indexes were in error, a result handle is returned by EhwSearch to allow subsequent error handling by the caller.

Example

This is an example of an EhwGetProblemInfo function call:

```
/*-------------------------------------------------------------*/
/* invoke API function to obtain the index RC information      */
/*-------------------------------------------------------------*/
ulReturnCode =
EhwGetProblemInfo( pSession,         /* In  -- session pointer   */
                   ulResultHandle,   /* In  -- index handle      */
                   &ulDataLength,    /* Out -- data stream length*/
                   &pDataStream,     /* Out -- problem info      */
                   &ulDiagnosisInfo); /* Out -- diagnosis info.  */

if (ulReturnCode  != RC_DONE)        /* check the API return code*/
{
                                     /* handle the function error*/
}                                    /* endif API call failed    */
/*-------------------------------------------------------------*/
```

After completion of the function call, the area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would then have a value of 30.

```
Symbolic notation               Hexadecimal representation

05 IRX  S                       0005 0032 E2
13 XNAM A EHWINDEX              000D 003C C1 454857494E444558
07 XRC  A INDEX_NOT_ACCESSIBLE  0007 003E C1 02BC
05 IRX  E                       0005 0032 C5
```

EhwGetResultView

This function lists descriptive information for all documents of a specified search result list.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

result list handle
Identifies the search result list to be accessed. It is a handle returned by EhwCreateResultView or EhwSelectResultView.

Output parameters

data stream length
The length, in bytes, of the data contained in the *result list* parameter.

result list
A list of descriptions for all documents of the specified search result list. This parameter is returned in a data stream format (see “Data stream syntax”).

diagnosis information
There is no diagnosis information returned for this call.

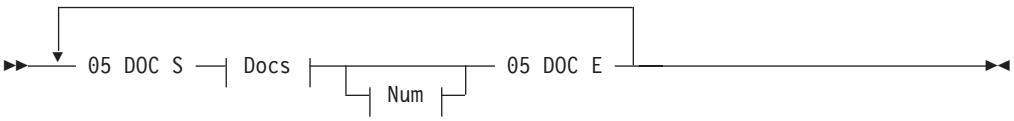
return code
The following return code values can be returned with this call:

RC_DONE
RC_CONTINUATION_MODE_ENTERED
RC_EMPTY_LIST
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_REQUEST_IN_PROGRESS
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_NOT_ENOUGH_MEMORY
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

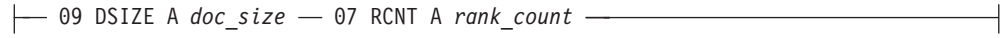
Result list



Docs:

Num:

Basic values:



The data stream items are:

DOC
Delimits the information returned for each document in the search result.

DID
Specifies the identifier of a document.

XNAM
Name of the index that the document belongs to.

RVAL
Rank value. This value is available only if EhwRank was used to add rank information to each document's result handle.

DSIZE
Number of word occurrences in the document, excluding stop words. This value is available only if EhwRank was used to add rank information to each document's result handle. It is not delivered for queries containing a free-text argument.

RCNT
The number of matches in the document that contributed to ranking.

This value is available only if EhwRank was used to add rank information to each document's result handle. It is not delivered for queries containing a free-text argument.

Usage

EhwGetResultView returns one portion of the result list. The size of each portion is determined by Text Search Engine and may be different for each call.

When the return code RC_CONTINUATION_MODE_ENTERED is returned, there may be more information to list. When the return code RC_DONE is returned, there is no more information available.

You can list only one search result at a time. If an EhwGetResultView call returns RC_CONTINUATION_MODE_ENTERED, you can continue listing the same result with further EhwGetResultView calls; but when you issue an EhwGetResultView call for a different result, the continuation mode of the previous call is canceled.

Note that EhwGetResultView does not check whether the users of your application are authorized to access the documents listed in a search result.

Example

This is an example of an EhwGetResultView function call:

```
/*--------------------------------------------------------------------*/
/* Call API function EhwGetResultView until end of data is indicated  */
/*--------------------------------------------------------------------*/
do
{
   ulReturnCode =
   EhwGetResultView (pSession,          /* In  -- session pointer     */
                     ulResultViewHandle,/* In  -- result view handle  */
                     &ulDataLength,     /* Out -- data stream length  */
                     &pDataStream,      /* Out -- result view data    */
                     &ulDiagnosisInfo); /* Out -- diagnosis info.     */
                                        /* check the API return code  */
   if ((ulReturnCode != RC_DONE) &&;
       (ulReturnCode != RC_CONTINUATION_MODE_ENTERED))
   {
                                   /* handle the function error     */
   }                               /* endif API call failed         */

   /*-----------------------------------------------------------------*/
   /* Parse the result view data stream ...                           */
   /*-----------------------------------------------------------------*/

}  while (ulReturnCode == RC_CONTINUATION_MODE_ENTERED)
/*--------------------------------------------------------------------*/
```

After completion of the last function call, the area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would then have a value of 76.

```
Symbolic notation             Hexadecimal representation

05 DOC  S                     0005 019A E2
31 DID  A \\NOE007\DATA\      001F 006A C1 5C5C4E...
          NEWS0815.DOC
13 XNAM A EHWINDEX            000D 003C C1 454857...
06 RVAL  50                   0006 0218 C1 32
09 DSIZE 2213                 0009 021A C1 0000 08A5
07 RCNT 2                     0007 0219 C1 0002
05 DOC  E                     0005 019A C5
```

EhwListDocumentModels

This function lists all document models.

The target of this call can be either the server instance related to the current session or any section-enabled index opened previously.

The set of models defined for the server instance serves as a collection of models to choose from when creating a section-enabled index. However, indexes can work with models unknown to the server instance. This can happen if a new model was defined for a specific index (see “EhwCreateDocumentModel” on page 49), but not for the server instance.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

index handle
Specifies the information-retrieval index to use. It is the handle returned by EhwOpenIndex when the index is opened. If this function is called to list all models defined for an index, this is the handle returned by a previous call to EhwOpenIndex. If this function is called to list all models defined for a server instance, the index handle must have the value 0L.

Output parameters

data stream length
The length, in bytes, of the data contained in the *document model list* datastream.

document model list
A list of all models defined for either the server instance or the index specified by the *index handle* input parameter. The output is returned in datastream format. See “Data stream syntax” on page 100.

diagnosis information
There is no diagnosis information returned for this call.

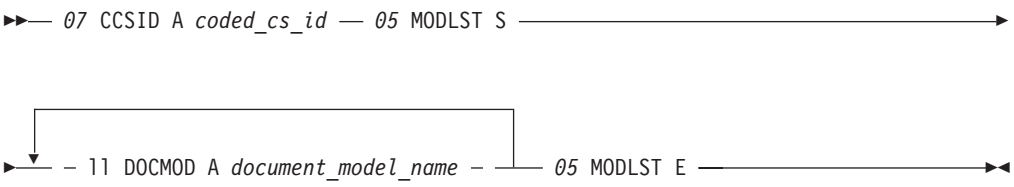
return code
The following return code values can be returned with this call:

RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_DATASTREAM_SYNTAX_ERROR
RC_INCORRECT_AUTHENTICATION (AIX only)
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_NOT_ENOUGH_MEMORY
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_CONFLICT_WITH_INDEX_TYPE
RC_DOCMOD_READ_PROBLEM
RC_INCORRECT_HANDLE

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Document model list



The data stream items are:

CCSID
Specifies the SAA Coded Character Set Identifier for the subsequent strings.

Depending on operating system, the values are EHWCCSID_00819 or EHWCCSID_00500.

MODLST
Delimits the list of documetn model names defined for the server instance or for the index specified by the index handle.

DOCMOD
A document model name in the document model list.

Usage

When creating a new index that is to support sections, use this call to get a list of available document models on your server instance. If creation of an index failed with RC_DOCMOD_READ_PROBLEM, use this call (index handle 0L) to get a list of the available document models for the server instance you work with. If queries fail with RC_DOCMOD_READ_PROBLEM, use this call, with an index handle for the index that was used for the query, to get a list of document models defined for the index you want to search on. If Update fails to index certain documents, indicating only ″UNKNOWN_DOCUMENT_MODEL″ when EhwGetIndexingMessages is called, use this call, with an index handle for the index which was used for update, to get a list of document models defined for that index. To get information on a specific document model, use the API call “EhwGetDocumentModel” on page 72. For information on which default model is used for a specific index, use “EhwGetIndexingRules” on page 84.

Example

This is an example of an EhwListDocumentModels function call:

```
/*--------------------------------------------------------------------*/
/* List document models known to a information-retrieval index        */
/*--------------------------------------------------------------------*/
ulReturnCode =
EhwListDocumentModels (pSession,           /* In  -- session pointer   */
                       ulIndexHandle,      /* IN  -- index_handle      */
                       &ulDataLength,      /* OUT -- data stream length*/
                       &pDataStream,       /* OUT -- index information */
                       &ulDiagnosisInfo);  /* Out -- diagnosis info.   */

if (ulReturnCode != RC_DONE)            /* check the API return code   */
{
```

```
                                        /* handle the function error   */
}                                       /* endif API call failed       */
/*-------------------------------------------------------------------------*/
```

EhwListIndexes

This function provides a list of the information-retrieval indexes accessible from the current session.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

Output parameters

data stream length
The length, in bytes, of the data contained in the *index list* parameter.

index list
A list of the information-retrieval indexes accessible from the current session. This parameter is returned in a data stream format (see “Data stream syntax”).

diagnosis information
There is no diagnosis information returned for this call.

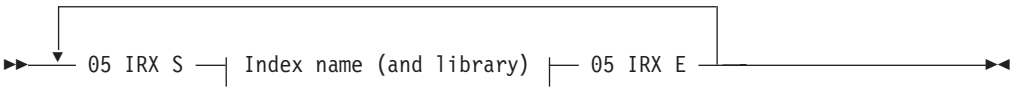
return code
The following return code values can be returned with this call:

RC_DONE
RC_CONTINUATION_MODE_ENTERED
RC_EMPTY_LIST
RC_UNKNOWN_SESSION_POINTER
RC_UNKNOWN_SERVER_NAME
RC_REQUEST_IN_PROGRESS
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_NOT_ENOUGH_MEMORY
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR

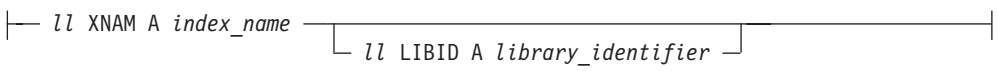
See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Index list



Index name (and library):



The data stream items are:

IRX
Delimits the information returned for each index.

XNAM
Specifies the name of the information-retrieval index.

LIBID
Specifies the library identifier of the information-retrieval index.

Usage

If provided, the LIBID item has to be specified as an input parameter when the library services function LIB_init is called.

Example

This is an example of an EhwListIndexes function call:

```
/*-------------------------------------------------------------------*/
/* Call API function EhwListIndexes until end of data is indicated */
/*-------------------------------------------------------------------*/
do
{
   ulReturnCode =
   EhwListIndexes (pSession,        /* In  -- session pointer      */
                   &ulDataLength,   /* Out -- data stream length   */
                   &pDataStream,    /* Out -- index list           */
                   &ulDiagnosisInfo /* Out -- diagnosis info.      */
                  );
                                    /* check the API return code   */
   if ((ulReturnCode != RC_DONE) &&;
       (ulReturnCode != RC_CONTINUATION_MODE_ENTERED))
   {
                                    /* handle the function error   */
   }                                /* endif API call failed       */

   /*-----------------------------------------------------------------*/
   /* Parse the index list data stream ...                            */
   /*-----------------------------------------------------------------*/

}  while (ulReturnCode == RC_CONTINUATION_MODE_ENTERED)
/*-------------------------------------------------------------------*/
```

After completion of the function call, the area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would then have a value of 57.

```
Symbolic notation              Hexadecimal representation

05 IRX   S                     0005 0032 E2
13 XNAM  A EHWINDEX            000D 003C C1 454857494E444558
05 IRX   E                     0005 0032 C5
05 IRX   S                     0005 0032 E2
12 XNAM  A INDEX07             000C 003C C1 494E4445583037
12 LIBID A LIBID07             000C 0041 C1 4C494249443037
05 IRX   E                     0005 0032 C5
```

EhwListResult

This function lists the documents of a specified search result. Use this function when you are not interested in working with a search result view but simply want a list of all documents resulting from an EhwSearch request.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

result handle
Identifies the search result to be listed. It is the handle returned by an EhwSearch call.

Output parameters

data stream length
The length, in bytes, of the data contained in the *result list* parameter.

result list
A list of the documents of the specified search result. If the result contains documents from more than one index, the function lists the documents of the search result and the index name where each of them was found. This parameter is returned in a data stream format (see “Data stream syntax”).

diagnosis information
There is no diagnosis information returned for this call.

return code
The following return code values can be returned with this call:

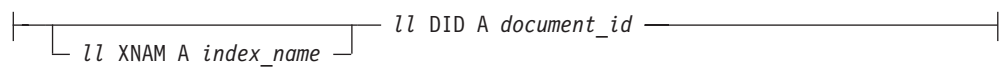
RC_DONE
RC_CONTINUATION_MODE_ENTERED
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_REQUEST_IN_PROGRESS
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_NOT_ENOUGH_MEMORY
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Result list

Doc info:



The data stream items are:

DOC
Delimits the information returned for each document in the search result.

XNAM
Specifies the name of the index where the document was found.

DID
Specifies the identifier of a document.

Usage

EhwListResult returns a portion of the result list. The size of each portion is determined by Text Search Engine and can be different for each call.

When the return code RC_CONTINUATION_MODE_ENTERED is returned, there may be more information to list. When the return code RC_DONE is returned, there is no more information available.

You can list only one search result at a time. If an EhwListResult call returns RC_CONTINUATION_MODE_ENTERED, you can continue listing the same result with further EhwListResult calls; but when you issue an EhwListResult call for a different result, the continuation mode of the previous call is canceled.

Note that EhwListResult does not check whether the users of your application are authorized to access the documents listed in a search result.

Example

This is an example of an EhwListResult function call:

```
/*-------------------------------------------------------------------*/
/* Call API function EhwListResult until end of data is indicated  */
/*-------------------------------------------------------------------*/
do
{
   ulReturnCode =
   EhwListResult (pSession,         /* In  -- session pointer      */
                  ulResultHandle,   /* In  -- result handle        */
                  &ulDataLength,    /* Out -- data stream length   */
                  &pDataStream,     /* Out -- result list          */
                  &ulDiagnosisInfo);/* Out -- diagnosis info.      */
                                    /* check the API return code   */
   if ((ulReturnCode != RC_DONE) &&;
       (ulReturnCode != RC_CONTINUATION_MODE_ENTERED))
   {
                                    /* handle the function error   */
   }                                /* endif API call failed       */

   /*----------------------------------------------------------------*/
   /* Parse the result list data stream ...                        */
   /*----------------------------------------------------------------*/

}  while (ulReturnCode == RC_CONTINUATION_MODE_ENTERED)
/*-------------------------------------------------------------------*/
```

ListResult

After completion of the last function call, the area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would then have a value of 163.

```
Symbolic notation              Hexadecimal representation

05 DOC  S                      0005 019A E2
31 DID  A \\NOE007\DATA\       001F 006A C1 5C5C4E...
          NEWS0815.DOC
31 DID  A \\NOE007\DATA\       001F 006A C1 5C5C4E...
          NEWS2259.DOC
30 DID  A \\NOE099\LIB2\       001E 006A C1 5C5C4E...
          USGUIDE.DOC
30 DID  A \\NOE099\LIB2\       001E 006A C1 5C5C4E...
          HLPINFO.DOC
31 DID  A \\NOE099\LIB2\       001F 006A C1 5C5C4E...
          PLATFORM.DOC
05 DOC  E                      0005 019A C5
```

EhwListServers

This function lists the symbolic names of the Text Search Engine servers connected to the client workstation on which it is called.

The symbolic name of a server is the search service name provided when configuring or creating a client. Each search service is connected to one server; its name can be used as input to an EhwStartSession call.

Input parameters

None.

Output parameters

data stream length
The length, in bytes, of the data contained in the *server list* parameter.

server list
A list of the Text Search Engine servers that can be connected to the calling application. This parameter is returned in a data stream format (see “Data stream syntax”).

diagnosis information
There is no diagnosis information returned for this call.

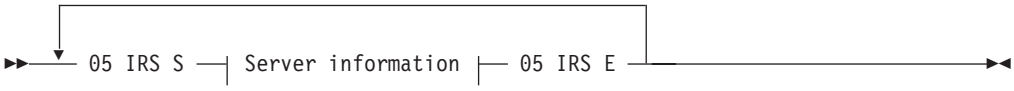
return code
The following return code values can be returned with this call:

RC_DONE
RC_EMPTY_LIST
RC_UNEXPECTED_ERROR
RC_NOT_ENOUGH_MEMORY
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR

See also “Appendix A. The API return codes” on page 191.

Data stream syntax

Server list



Server information

The data stream items are:

IRS
Delimits the information returned for each server.

IRSN
Specifies the symbolic name of the Text Search Engine server. This name is associated with the server when the service object is defined on the client workstation. It must be specified when you want to start a session with this server.

IRSL
Indicates the location of the Text Search Engine server. The *server location* is a 1-byte code that can have one of the following values:

IRSL_LOCAL
for a local server on the client workstation.

IRSL_REMOTE
for a remote server on a LAN workstation.

The output of a server location refers to the communication protocol chosen to be used between the search service and the server, rather than the exact physical location of the search service and the server. Whenever the communication protocol is 'local', the server location is `IRSL_LOCAL`. Whenever the communication protocol is 'TCPIP', the server location is `IRSL_REMOTE`, even if the search service and the server are physically on the same machine.

Usage

The servers are listed regardless of the status of each server or whether a connection can be established at this time. Availability of the server is checked only when you actually start an information-retrieval session using EhwStartSession.

Example

This is an example of an EhwListServers function call:

```
/*--------------------------------------------------------------------*/
/*  Obtain a list of the search servers                               */
/*--------------------------------------------------------------------*/
  ulReturnCode =
  EhwListServers (&ulDataLength,      /* Out -- data stream length   */
                  &pDataStream,       /* Out -- server list          */
                  &ulDiagnosisInfo);  /* Out -- diagnosis info.      */

  if (ulReturnCode != RC_DONE)        /* check the API return code   */
  {
                                      /* handle the function error   */
  }                                   /* endif API call failed       */
/*--------------------------------------------------------------------*/
```

After completion of the function call, the area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would then have a value of 87.

```
Symbolic notation             Hexadecimal representation

05 IRS  S                     0005 000A E2
13 IRSN A SERVER01            000D 000B C1 5345525645523031
06 IRSL A IRSL_LOCAL          0006 000C C1 00
05 IRS  E                     0005 000A C5
05 IRS  S                     0005 000A E2
13 IRSN A SERVER02            000D 000B C1 5345525645523032
06 IRSL A IRSL_REMOTE         0006 000C C1 01
05 IRS  E                     0005 000A C5
05 IRS  S                     0005 000A E2
13 IRSN A SERVER03            000D 000B C1 5345525645523033
06 IRSL A IRSL_REMOTE         0006 000C C1 01
05 IRS  E                     0005 000A C5
```

EhwOpenDocument

This function prepares a document to return the document text and the highlighting information needed by a subsequent call of EhwGetMatches.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

result handle
Identifies the search result from which the document is to be browsed. It is the handle returned by an EhwSearch call.

data stream length
The length, in bytes, of the data contained in the *browse specifications* parameter.

browse specifications
Specifies the document to be browsed. This parameter is supplied in a data stream format (see “Data stream syntax” on page 111).

processing condition
Specifies whether the highlight information is to be obtained using a dictionary.
- MATCH_FAST: without dictionary
- MATCH_EXTENDED: with dictionary

For Ngram indexes, the processing condition is ignored.

Output parameters

document handle
Identifies the prepared document for later call of EhwGetMatches.

diagnosis information
There is no diagnosis information returned for this call.

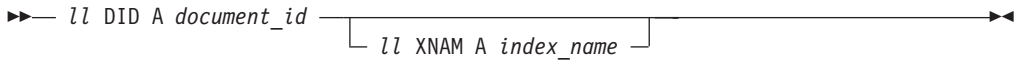
return code
The following return code values can be returned with this call:

RC_DONE
RC_DOCUMENT_IN_ERROR
RC_DOCUMENT_NOT_FOUND
RC_DOCUMENT_NOT_ACCESSIBLE
RC_UNKNOWN_SESSION_POINTER
RC_UNKNOWN_INDEX_NAME
RC_DATASTREAM_SYNTAX_ERROR
RC_DOCUMENT_IN_ERROR
RC_DOCUMENT_NOT_SUPPORTED
RC_REQUEST_IN_PROGRESS
RC_INCORRECT_HANDLE
RC_NOT_ENOUGH_MEMORY
RC_LS_NOT_EXECUTABLE
RC_LS_FUNCTION_FAILED
RC_IO_PROBLEM
RC_UNEXPECTED_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Browse specifications



The data stream items are:

DID
Specifies the identifier of the document whose text and the highlighting information is required.

XNAM
Specifies the name of the index to be used. This item must be provided for results of searches across several indexes. When the resulting documents are in only one index, this item can be omitted.

Usage

This function allows a user to prepare a document for later returning of the document text and highlighting information. The returned document handle is allowed only in subsequent calls of EhwGetMatches and EhwCloseDocument.

When the search service is integrated in a library system, such as IBM Content Manager, the document must be accessed in the library's document repository. If logging on to the library is required during this access, you must pass library information, such as user ID and password, at start session for use by EhwOpenDocument.

Example

This is an example of an EhwOpenDocument function call:

```
/*-------------------------------------------------------------------*/
/* Open the document to get the document text with matches           */
/*-------------------------------------------------------------------*/
ulReturnCode =
EhwOpenDocument   (pSession,        /* In  -- session pointer      */
                   ulResultHandle,  /* In  -- result handle        */
                   ulDataLength,    /* In  -- data stream length   */
                   pDataStream,     /* In  -- browse specifications*/
                   ProcMode,        /* In  -- processing mode      */
                   pulDocHandle, ,  /* Out -- document handle      */
                   pulDiagnosisInfo /* Out -- diagnosis info       */
                  );

if (ulReturnCode != RC_DONE)        /* check the API return code   */
{
                                    /* handle the function error   */
}                                   /* endif API call failed       */
/*-------------------------------------------------------------------*/
```

The area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would have a value of 51.

OpenDocument

```
Symbolic notation              Hexadecimal representation

31 DID  A \\NOE099\LIB2\       001F 006A C1 5C5C4E...
          PLATFORM.DOC
12 XNAM A EHWINDEX             000C 003C C1 454857...
```

EhwOpenIndex

This function opens an information-retrieval index. You must open an index before you can perform functions such as search or update.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

data stream length
The length, in bytes, of the data contained in the *index name* parameter.

index name
Specifies the information-retrieval index to be opened. This parameter is supplied in a data stream format (see “Data stream syntax” on page 114).

Output parameters

index handle
Identifies the information-retrieval index and its status. The index handle is used as an input parameter for search and indexing functions.

diagnosis information
There is no diagnosis information returned for this call.

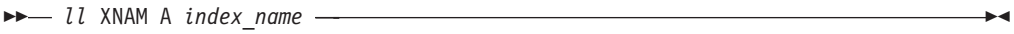
return code
The following return code values can be returned with this call:

RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_UNKNOWN_INDEX_NAME
RC_DATASTREAM_SYNTAX_ERROR
RC_REQUEST_IN_PROGRESS
RC_UNEXPECTED_ERROR
RC_INDEX_ALREADY_OPENED
RC_MAX_NUMBER_OF_OPEN_INDEXES
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_INDEX_NOT_ACCESSIBLE
RC_NOT_ENOUGH_MEMORY
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Index name



The data stream item is:

XNAM
Specifies the name of the index to be opened. It must be one of the names returned by the EhwListIndexes function for the current session.

Usage

You can open the same index only once per session.

Example

This is an example of an EhwOpenIndex function call:

```
/*-------------------------------------------------------------------*/
/* Call API function to open the information-retrieval index         */
/*-------------------------------------------------------------------*/
ulReturnCode =
EhwOpenIndex (pSession,             /* In  -- session pointer        */
              ulDataLength,         /* In  -- data stream length     */
              pDataStream,          /* In  -- index name             */
              &ulIndexHandle,       /* Out -- index handle           */
              &ulDiagnosisInfo);    /* Out -- diagnosis info.        */

if (ulReturnCode != RC_DONE)        /* check the API return code     */
{
                                    /* handle the function error     */
}                                   /* endif API call failed         */
/*-------------------------------------------------------------------*/
```

The area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would have a value of 13.

```
Symbolic notation              Hexadecimal representation

13 XNAM A EHWINDEX             000D 003C C1 454857494E444558
```

EhwRank

This function assigns a relevance value to each document of a result. The value indicates how relevant the document is to the query that produced the result. It is based on factors such as the number of search terms found in the document in relation to the number of documents. The relevance value is relative; that is, it indicates how relevant the document is in relation to the other found documents.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

result handle
Identifies the search result to apply ranking to. It is the handle returned by an EhwSearch call.

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

return code
The following return code values can be returned with this call:
RC_DONE
RC_EMPTY_LIST
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_UNEXPECTED_ERROR
RC_NO_RANKING_DATA_AVAILABLE
RC_NOT_ENOUGH_MEMORY
RC_RESULT_ALREADY_RANKED

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Usage

There is no output from EhwRank. Ranking applies to a search result including document data, index data, and query data. The relevance value depends on the structure of the query, the frequency of terms within a document, the size of documents, and the number of documents a term occurs in.

EhwRank assigns a relevance value to individual documents. To sort the list of documents according to this value, run EhwCreateResultView, followed by EhwSort. A relevance value is an unsigned char from hex 0 to hex 64. Note that a result can be ranked only if the `RANK` processing condition was used for the corresponding EhwSearch call. Otherwise, the EhwRank function will return an RC_NO_RANKING_DATA_AVAILABLE return code.

The next two sections explain the ranking methods used for Boolean and free-text or hybrid queries.

Boolean queries

For Boolean queries, the following ranking method is used to assign a relevance value to a document of a result:

1. Single word search terms contribute to a document's relevance value through the ratio of their occurrence frequency to the number of documents they occur in.

   This implies the following:

   a. Frequent occurrence of a search term increases a document's relevance value.
   b. The fewer documents are found for a term, the more relevant is a document the term occurs in.
   c. A more restrictive query expression usually contributes more to the relevance value of a document than a less restrictive one.
2. Different queries with logically the same content can produce the same or a very similar rank sequence for the same document collection. For example, `(term1 OR term2) AND term3` is logically equivalent to `(term1 AND term3) OR (term2 AND term3)` and therefore produces the same, or nearly the same, rank sequence.
3. The more synonyms there are, the less each of them adds to the relevance value. So the sum of the single relevance values of synonyms of a term is less than or equal to the relevance value of the term itself.

Note that it is only the ordering of relevance values that remains stable for logically equivalent queries, not necessarily the values themselves.

Furthermore, relevance values and relevance orderings obtained from different indexes cannot be compared. Therefore, different applications of EhwSearch against different indexes or index groups usually produce results that cannot be sorted by relevance value or merged according to relevance ordering. However, relevance values obtained from one cross-index search can be compared across indexes, of course, because in that case the index group is treated as a single index. Note that even when relevance ordering or values cannot be used, the so-called rank count can, because this is a sum of matches over documents that is independent both from query structure and the index content.

Free-text and hybrid queries

A query containing both a Boolean query and a free-text argument is called a *hybrid* query. For free-text and hybrid queries, a relevance value is assigned to each document that contains at least one of the nonstopwords in the free-text argument.

Words that occur close to each other in the text form *lexical affinities*. The more often a document contains an individual word or a lexical affinity, the higher the relevance value. Terms that are rare in the collection of documents as a whole are scored higher than common terms. Scores greater than, say, 85 generally indicate very good matches; scores less than about 15 generally indicate poor matches.

The result of a hybrid query is ranked according to the ranking scheme of the free-text part of the query only.

Example

This is an example of an EhwRank function call:

```
/*-------------------------------------------------------------------*/
/* call API function EhwRank                                         */
/*-------------------------------------------------------------------*/
   ulReturnCode =
            EhwRank (
                  pSession,         /* In  -- session pointer      */
                  ulResultHandle,   /* In  -- result handle        */
```

```
                &ulDiagnosisInfo);/* Out -- diagnosis info.      */
                                  /* check the API return code   */
   if (ulReturnCode != RC_DONE)
   {
                                  /* handle the function error   */
   }                              /* endif API call failed       */


/*-------------------------------------------------------------------*/
```

EhwReorgIndex

The reorganization function removes obsolete information from the index, and compresses the information to improve storage utilization and performance. For details on why and when an index should be reorganized, refer to “Improving performance and the use of storage” on page 25.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

index handle
Identifies the information-retrieval index to be reorganized. It is the handle returned by EhwOpenIndex when the index is opened.

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

return code
The following return code values can be returned with this call:

RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_REQUEST_IN_PROGRESS
RC_INCORRECT_AUTHENTICATION (Text Search Engine for AIX only)
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_CONFLICTING_TASK_RUNNING
RC_NOT_ENOUGH_MEMORY
RC_NO_ACTION_TAKEN
RC_FUNCTION_DISABLED
RC_FUNCTION_IN_ERROR
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR
RC_MAX_NUMBER_OF_TASKS
RC_GTR_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Usage

EhwReorgIndex starts the reorganization of the specified information-retrieval index. After checking whether the index can be reorganized and setting up the reorganization process, EhwReorgIndex returns control to your application. The return code RC_DONE signals that the reorganization task has been started successfully; your application is not notified when and if this task has completed. Use EhwGetIndexFunctionStatus to check for the correct processing of administration tasks. If EhwGetIndexFunctionStatus returns an error code, refer to “Appendix B. Error codes returned by GetIndexingMsgs and GetIndexFunctionStatus” on page 207.

Example

This is an example of an EhwReorgIndex function call:

```
/*-------------------------------------------------------------------*/
/* Call EhwReorgIndex to start a reorganize index task               */
/*-------------------------------------------------------------------*/
ulReturnCode =
EhwReorgIndex (pSession,            /* In  -- session pointer      */
               ulIndexHandle,       /* In  -- index handle         */
               pulDiagnosisInfo);   /* Out -- diagnosis info       */

if (ulReturnCode != RC_DONE)        /* check the API return code   */
{
                                    /* distinguish the code types  */
   /*----------------------------------------------------------------*/
   /*                                  no error code:                */
   /* case RC_NO_ACTION_TAKEN:         no active secondary index     */
   /*                                  ...                           */
   /* case RC_CONFLICTING_TASK_RUNNING:                              */
   /*                                  active update | reorg. task */
   /*                                  ...                           */
   /* case RC_FUNCTION_DISABLED:       update function is disabled */
   /*                                  ...                           */
   /*                                  continue with processing      */
   /*----------------------------------------------------------------*/
   /*                                  function error:               */
   /* case RC_UNEXPECTED_ERROR:                                      */
   /* case RC_SERVER_BUSY:                                           */
   /* case RC_SERVER_CONNECTION_LOST:                                */
   /* case RC_COMMUNICATION_PROBLEM:                                 */
   /* case RC_NOT_ENOUGH_MEMORY:                                     */
   /* case RC_FUNCTION_IN_ERROR:                                     */
   /* case RC_IO_PROBLEM:                                            */
   /* case RC_WRITE_TO_DISK_ERROR:                                   */
   /*                                  handle the function error     */
   /*                                  ...                           */
   /*                                  stop processing / return      */
   /*----------------------------------------------------------------*/
   /* default:                         application error:            */
   /*                                  handle the internal error     */
   /*                                  ...                           */
   /*                                  stop processing / return      */
   /*----------------------------------------------------------------*/
}                                   /* endif API return code > 0     */
/*-------------------------------------------------------------------*/
```

EhwResumeIndex

This function resumes a suspended information-retrieval index so that it can be opened.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

data stream length
The length, in bytes, of the data contained in the *index name* parameter.

index name
Specifies the information-retrieval index to be resumed. This parameter is supplied in a data stream format (see “Data stream syntax” on page 121).

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

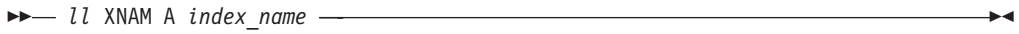
return code
The following return code values can be returned with this call:

RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_UNKNOWN_INDEX_NAME
RC_DATASTREAM_SYNTAX_ERROR
RC_REQUEST_IN_PROGRESS
RC_INCORRECT_AUTHENTICATION (Text Search Engine for AIX only)
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_INDEX_NOT_ACCESSIBLE
RC_NOT_ENOUGH_MEMORY
RC_NO_ACTION_TAKEN
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Index name



The data stream item is:

XNAM
Specifies the name of the index to be resumed. It must be one of the names returned by the EhwListIndexes function for the current session.

Usage

The EhwResumeIndex function cancels the effect of a previous EhwSuspendIndex function call.

Example

This is an example of an EhwResumeIndex function call:

```
/*-------------------------------------------------------------------*/
/* Call API function to resume the information-retrieval index     */
/*-------------------------------------------------------------------*/
ulReturnCode =
EhwResumeIndex (pSession,           /* In  -- session pointer      */
                ulDataLength,       /* In  -- data stream length   */
                pDataStream,        /* In  -- index name           */
                &ulDiagnosisInfo);  /* Out -- diagnosis info.      */

if (ulReturnCode != RC_DONE)        /* check the API return code   */
{
                                    /* handle the function error   */
}                                   /* endif API call failed       */
/*-------------------------------------------------------------------*/
```

The area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would have a value of 12.

```
Symbolic notation             Hexadecimal representation

12 XNAM A INDEX07             000C 003C C1 494E4445583037
```

EhwScheduleDocument

This function schedules documents for indexing or for deletion from an information-retrieval index.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

index handle
Identifies the information-retrieval index to which you want to add, or from which you want to delete, documents. It is the handle returned by EhwOpenIndex when the index is opened.

data stream length
The length, in bytes, of the data contained in the *scheduling list* parameter.

scheduling list
Lists the documents to be scheduled and specifies whether they are to be added to the index or deleted from the index. This parameter is supplied in a data stream format (see “Data stream syntax” on page 123).

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

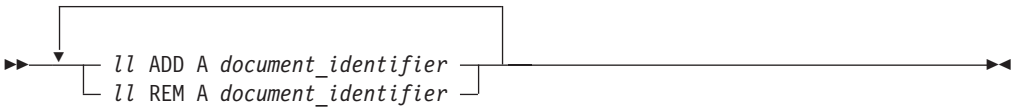
return code
The following return code values can be returned with this call:

RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_DATASTREAM_SYNTAX_ERROR
RC_REQUEST_IN_PROGRESS
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_NOT_ENOUGH_MEMORY
RC_FUNCTION_DISABLED
RC_FUNCTION_IN_ERROR
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Scheduling list



The data stream items are:

ADD
Specifies a document to be indexed or re-indexed. The request is scheduled (see “Usage”) and is later processed as follows:

- If the specified document exists and can be accessed by the Text Search Engine server, its indexing information is added to the specified index.
- If the index already contains indexing information for this particular document, the existing information is replaced.
- If the specified document does not exist or cannot be accessed by the Text Search Engine server, the request is ignored.

REM
Specifies a document whose indexing information is to be removed from the specified information-retrieval index. The request is scheduled (see “Usage”) and is later processed as follows:

- If the index contains indexing information for this particular document, it is removed.
- If the index does not contain indexing information for this particular document, the request is ignored.

Usage

This function puts the ADD or REM requests into an indexing queue that is maintained for each information-retrieval index. The actual indexing of documents and modification of the information-retrieval index take place when the EhwUpdateIndex function is processed.

When you specify several ADD and REM requests in a single EhwScheduleDocument call, either all the requests are scheduled (return code RC_DONE) or none of the requests are scheduled (all other return codes).

Example

This is an example of an EhwScheduleDocument function call:

```
/*-------------------------------------------------------------------*/
/* Issue an API function call to schedule the documents              */
/*-------------------------------------------------------------------*/
ulReturnCode = EhwScheduleDocument
               (
                pSession,           /* In  -- session pointer      */
                ulIndexHandle,      /* In  -- index handle         */
                ulDataLength,       /* In  -- data stream length   */
                pDataStream,        /* In  -- scheduling list      */
                pulDiagnosisInfo    /* Out -- diagnosis info       */
               );

if (ulReturnCode != RC_DONE)        /* check the API return code   */
```

```
   {
                                     /* handle the function error   */
   }                                 /* endif API call failed       */
   /*-------------------------------------------------------------------*/
```

The area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would have a value of 217.

```
Symbolic notation              Hexadecimal representation

31 ADD  A \\NOE002\DATA\       001F 01FE C1 5C5C4E...
          NEWS1020.DOC
31 ADD  A \\NOE002\DATA\       001F 01FE C1 5C5C4E...
          NEWS0207.DOC
31 ADD  A \\NOE002\DATA\       001F 01FE C1 5C5C4E...
          NEWS1022.DOC
31 ADD  A \\NOE002\DATA\       001F 01FE C1 5C5C4E...
          NEWS1030.DOC
31 REM  A \\NOE007\DATA\       001F 0208 C1 5C5C4E...
          NEWS2259.DOC
31 REM  A \\NOE099\LIB2\       001F 0208 C1 5C5C4E...
          PLATFORM.DOC
31 ADD  A \\NOE002\DATA\       001F 01FE C1 5C5C4E...
          NEWS1021.DOC
```

Here is an example datastream for scheduling documents in MVS™ datasets:

```
29 ADD  A \\'SYS1.PARMLIB (MEMBER)'
```

EhwSearch

This function searches for the information that you specify in the query parameter. It searches either in one information-retrieval index or in multiple indexes belonging to an index group created by EhwCreateIndexGroup. For an index group, the query scope information has to be supplied separately by EhwAddQueryScope.

If your query contains a free-text argument in addition to, or instead of, a Boolean query, you get a result *per index*.

This means that free-text rank values for documents are calculated for each index; a document contained in two indexes may show different rank values. Note, however, that ranking across several indexes is supported for Boolean queries.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

index handle or index group handle
Identifies the handle of the index or index group to be searched.

- *index handle* identifies the information-retrieval index to search. It is the handle returned by EhwOpenIndex when the index is opened.
- *index group handle* is the handle returned by EhwCreateIndexGroup.

data stream length
The length, in bytes, of the data contained in the *query* parameter.

query
The information that you want to find, and the criteria for finding it. This parameter is supplied in a data stream format (see “Data stream syntax” on page 126).

Output parameters

result handle
Identifies the search result, that is, the set of documents found that comply with the query specifications. The result handle is an input parameter in function calls that list or delete the search result. It is also an input parameter for CreateResultView which creates a view on the search result.

result size
The number of documents found with this search request. If the result size is zero, no result handle is normally returned. If, however, the result size is zero and the search was done on multiple indexes (cross-index search), RC_INDEX_GROUP_SEARCH_ERROR is returned if one or more of the indexes did not contribute to the result (index-specific error or index disabled or index suspended). In that case, a result handle is returned so that EhwGetProblemInfo can be called.

diagnosis information
Returned only with the return code RC_DATASTREAM_SYNTAX_ERROR. It contains an offset value that points to a data stream item in the *query* parameter that caused the error. For example, the item may be out of sequence, it may have an unknown identifier or type, or it may contain an invalid value.

return code

The following return code values can be returned with this call:

RC_DONE
RC_DICTIONARY_NOT_FOUND
RC_STOPWORD_IGNORED
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_CCS_NOT_SUPPORTED
RC_LANGUAGE_NOT_SUPPORTED
RC_CONFLICT_WITH_INDEX_TYPE
RC_INVALID_MASKING_SYMBOL
RC_DATASTREAM_SYNTAX_ERROR
RC_QUERY_TOO_COMPLEX
RC_PROCESSING_LIMIT_EXCEEDED
RC_INDEX_GROUP_SEARCH_ERROR
RC_INDEX_SPECIFIC_ERROR
RC_REQUEST_IN_PROGRESS
RC_UNEXPECTED_ERROR
RC_MAX_NUMBER_OF_RESULTS
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_NOT_ENOUGH_MEMORY
RC_EMPTY_QUERY
RC_EMPTY_INDEX
RC_FUNCTION_DISABLED
RC_FUNCTION_IN_ERROR
RC_INSTALLATION_PROBLEM
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR
RC_GTR_ERROR
RC_CONFLICT_WITH_INDEX_TYPE
RC_UNKNOWN_SECTION_NAME
RC_DOCMOD_READ_PROBLEM

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

In a query, you specify criteria for searching the textual content of indexed documents. The search criteria can contain a Boolean query, or a free-text argument or both. If both a Boolean query and a free-text argument are specified, the Boolean query must be specified first. The Boolean query and the free-text argument are implicitly connected by an AND operator.

You can connect several text criteria within a Boolean query using the Boolean operators AND and OR. EhwSearch supports full Boolean search on all index types. In addition, queries requesting documents to contain search terms in a specified proximity are supported.

Search terms can be expanded using synonyms, or thesaurus terms. For Ngram indexes, search terms can be requested to match document text within a certain threshold. You can also specify processing conditions to cancel long-running queries or to truncate large results and a query scope to restrict the search to a group of indexed documents.

For a cross-index search, you must supply the complete query scope separately using EhwAddQueryScope. These scope settings are valid only for the duration of a search.

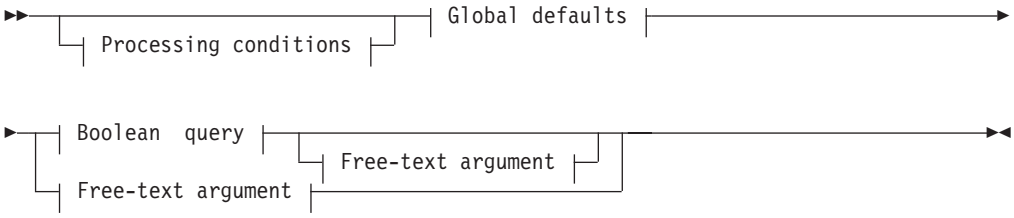
You must specify the CCSID and national language of search terms in the query. These are the default values for all search terms in the query. Each search term can be specified to have its own CCSID and national language.

For queries on an Ngram index, the CCSID must be the same as the one the index was created for.

Note that some options are not supported if your query contains a free-text argument or if a search is to be performed on an Ngram index. See the description of these parameters.

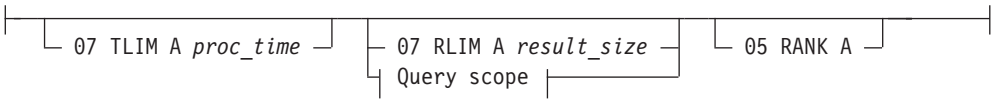
Query input parameter

Here is the data stream syntax for the query input parameter:

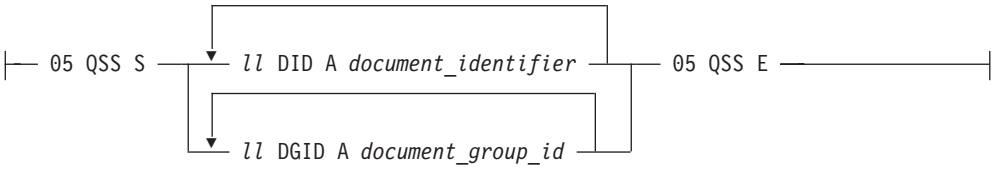


The “Processing conditions” let you specify how the query should be processed. The “Global defaults” on page 128 define the default settings used for processing the query. Some of these values may be overwritten by local settings. The “Boolean query” on page 129 lets you construct queries using full Boolean operations. The “Free-text argument” on page 136 lets you make a query in natural language. A free-text query processes the words in the query and finds the best-matching documents. It does not require all the words to occur in a found document, but the rank value it returns corresponds to the number of hits.

Processing conditions



Query scope:



For the positioning of processing conditions, see “Query input parameter”.

The data stream items are:

TLIM
Specifies the maximum processing time of the Text Search Engine server for a Boolean query or the Boolean part of a hybrid query. For a free-text-only query the TLIM processing condition has no effect. The same applies to search requests on an Ngram index. If the maximum processing time is exceeded, the search request is canceled, and no search result is built.

The processing time is specified in seconds as a 2-byte binary value in big-endian format. The value can be from 1 to 1440.

RLIM
Specifies the maximum result size for this search request. The *result size* is a 2-byte binary value in big-endian format. The value can be from 1 to 32767.

If the maximum result size is exceeded for a Boolean query, the number of documents as specified with the RLIM parameter are returned. For Boolean searches, the qualifying documents are in random order. They are not the subset yielding the highest rank scores. For cross-index search, RLIM is applied for each index; that is, the effective count of returned documents may be the number of indexes times RLIM.

If the maximum result size is exceeded for a query containing a free-text argument, the number of documents as specified with the RLIM parameter are returned. These documents have the highest rank scores.

The default RLIM value for a free-text or hybrid query is 50.

RANK
This flag requests that ranking data is provided by the server. The EhwRank function can be used only on results that have been produced by a query in which the flag was set.

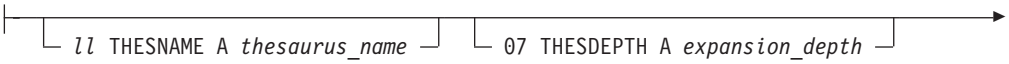
QSS
Delimits the scope of the query. If you specify a query scope, the search result is limited to the set of documents defined by the scope. The query scope is either a list of document identifiers or a list of document group identifiers.

DID
Specifies the identifier of a document that is indexed in the information-retrieval index to be searched.

DGID
Specifies a document group identifier, such as a directory name in a UNIX file system.

Note: The specification of document groups as a query scope assumes the naming conventions of file systems where document identifiers (fully qualified file names) always start with the document group identifier (subdirectory name). You should not use this feature with library systems that have different naming conventions.

Global defaults



For the positioning of global defaults, see “Query input parameter” on page 127.

The data stream items are:

THESNAME
Specifies the name of a thesaurus dictionary which is to be used to expand query terms. The default for the thesaurus name is `imothes` for indexes of type XTYP_LINGUISTIC and XTYP_PRECISE. For Ngram indexes, the default for the thesaurus name is `imonthes`. The default path is the servers resource directory as specified at installation. The language used for `imothes` and `imonthes` is English.

The sample thesauri are not considered as appropriate for a real search application. A search application can generate its own default thesaurus in the resource directory by using the same name. You can define your own thesauri. To be able to use them in a search application, you have to compile them using one of the thesaurus compilers provided. Refer to “Thesaurus concepts” on page 10 for definition formats and use of the thesaurus compilers. If the thesaurus name is fully qualified, the file must be available on the server machine.

THESDEPTH
Specifies the depth which is to be used for query expansion by looking for matches in the thesaurus. Actual expansion of the query is requested by using the THES keyword for each search term. The default for depth is 1.

CCSID
Specifies the SAA Coded Character Set Identifier for the search terms in the current query except those search terms that have their own CCSID item. File IMOLANG.H, provided with Text Search Engine, defines symbolic names for the CCSIDs that are supported by Text Search Engine.

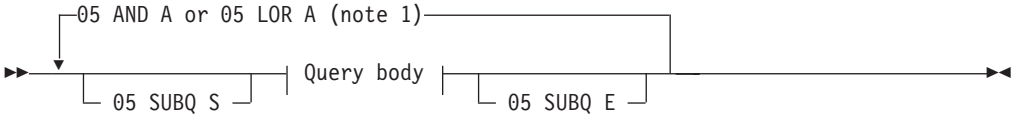
Do not use a CCSID that specifies a Unicode code page.

The 2-byte binary value must be specified in big-endian format.

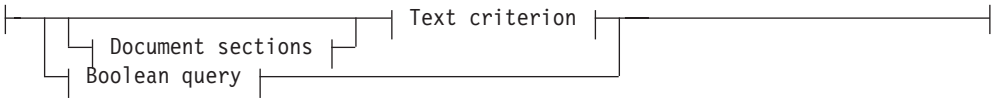
For a search on an Ngram index, the CCSID must be the same as the one the index was created for.

LANG
Specifies the language identifier for the search terms in the current query, except those search terms that have their own LANG item. File IMOLANG.H, provided with Text Search Engine, defines symbolic names for the language identifiers that are supported by Text Search Engine. The 2-byte binary value must be specified in big-endian format.

Boolean query



Query body:



Notes:

1. If you want to use several subqueries, connect them using the Boolean operator `05 AND A` or the Boolean operator `05 OR A` .

The data stream items are:

AND

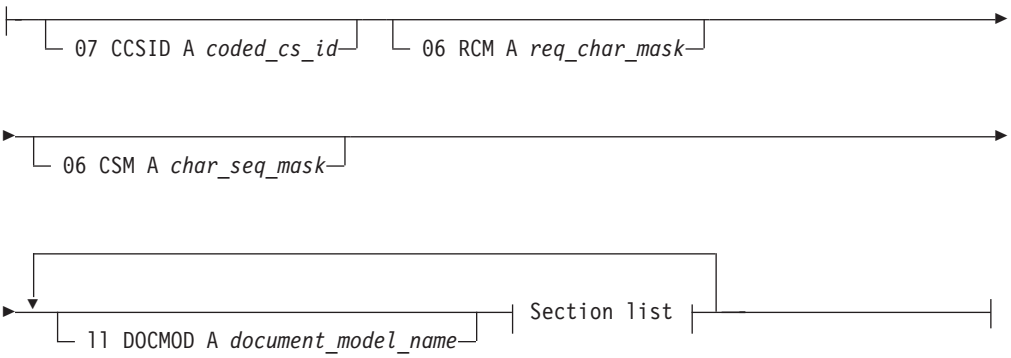
LOR
You can use these Boolean operators to connect several result subsets in a query.

If several Boolean operators are specified in a query, they are processed as follows: AND before LOR, left to right. To change the order of processing, use subquery notation: Boolean operators within subqueries are evaluated first.

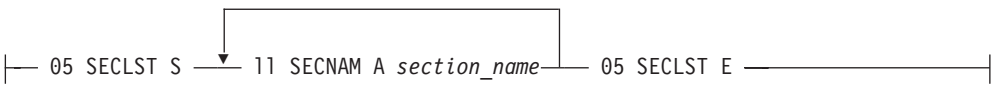
SUBQ
Delimits the start and end of a subquery. Use a subquery to change the default order of processing for the Boolean operators in a query.

For positioning of Boolean query, see “Query input parameter” on page 127. More complex data stream items are“Document sections” and “Text criterion” on page 132.

Document sections



Section list:



For the positioning of Document sections, see “Boolean query” on page 129 or “Free-text argument” on page 136.

The data stream items are:

CCSID
CCSID encoding of the subsequent strings for document model and section name. If a CCSID is not specified, the global default for CCSID is used.

RCM
Defines the SBCS masking symbol for a single required character used in the section name list or the document model name.

CSM
Defines the SBCS masking symbol for a sequence of optional characters or for a single optional word used in the section names list or the document model name.

DOCMOD
The name of the model that the section names list (see SECLST) is defined for. Regardless of the CCSID value, valid input characters for these strings are restricted to the range A-Z, a-z, 0-9.

If you do not specify the name of a document model, the default document model for the index is used.

SECLST
Delimits a list of section names. For Ngram indexes, only one entry is allowed in this list.

SECNAM

The name of the section as defined in the model definition file. The name must be valid for the model specified for this list. If the query is processed on multiple indexes then each index must support the model and section names specified in this list. Regardless of the CCSID value, valid input characters for the strings specifying section data are restricted to the range of A-Z, a-z, 0-9.

If multiple section names of attribute sections are specified (explicitly or by using wildcards), all sections in the list must be of the same type.

Syntax for section qualification: For documents in XML format, you can use a different syntax for formulating section specifications. This implies a set of shortcut notations based on the nesting levels of the document model.

For this purpose, in the section name list, use the special character ″/″ as a nesting level delimiter, and use ″..″ to require elements to be skipped.

Using this syntax assumes that section names (see “EhwCreateDocumentModel” on page 49) are synchronized to the nesting structure of elements defined for the XML document. This means that full ″path qualifiers″ as used for navigation in an XML document must have been used for the definition of section names.

In addition, the name of the XML root element must match the name of the document model and the name of the root section name.

So input for the section list can consist of:

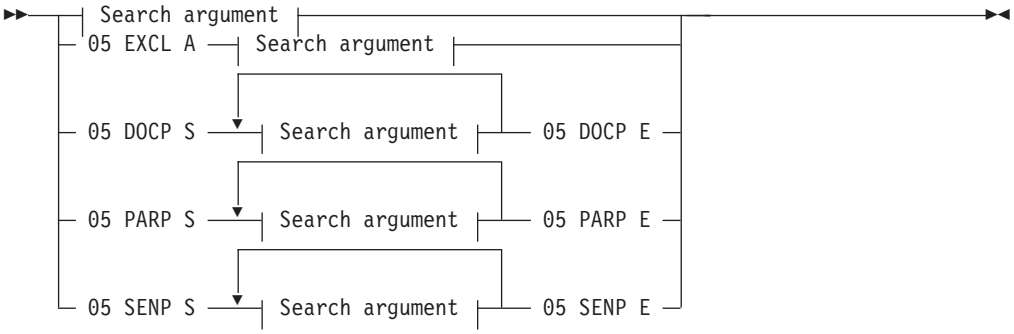
/ This represents the root element of an XML document. The query will look for text criterion in all document sections.

/elementName1/elementName2/.../elementNameX
This represents the element *elementNameX* somewhere in the navigational chain below *rootElement/elementName1/elementName2*. The number of elements specified before ″..″ can be any integer value starting from 1.

//elementNameX
This represents any element named *elementNameX* somewhere below the *rootElement*. *"//"* thus is a place holder for zero or more elements.

/elementName1//elementNameX
This represents any element named *elementNameX* somewhere in the navigational chain below *rootElement/elementName1*. Again, *"//"* is a place holder for zero or more elements.

Text criterion



For the positioning of text criterion, see “Boolean query” on page 129. A more complex data stream item is “Search argument”, which itself consists of one or more search terms.

The most simple text criterion consists of only one search argument.

The data stream items are:

EXCL
Expresses the Boolean operator NOT. You can use it to find the documents that do not contain the specified search argument.

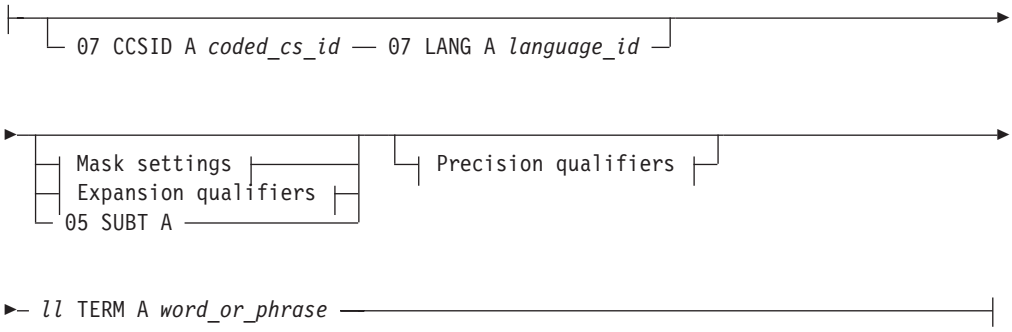
DOCP
Delimits a document proximity criterion. This is a sequence of two or more search arguments to be found within the same document.

PARP
Delimits a paragraph proximity criterion. This is a sequence of two or more search arguments to be found within the same paragraph of a document.

SENP
Delimits a sentence proximity criterion. This is a sequence of two or more search arguments to be found within the same sentence of a document.

Search argument

Search term:



For the positioning of search argument see “Text criterion” on page 132. You can also ask for query enhancement of each search term by using mask settings, expansion qualifiers, or precision qualifiers.

The most simple search argument consists of one search term. The most simple search term consists of one phrase or word. See the example for the datastream syntax of the most simple Boolean query.

The data stream items are:

SARG
Delimits the search argument. The SARG start and end items are required because you can specify several search terms in a single search argument. Where there are several search terms, they are treated as synonyms. That is, if any one of them is found in a document, the search argument is considered as met.

CCSID
Specifies the SAA Coded Character Set Identifier for the current search term. File IMOLANG.H, provided with Text Search Engine, defines symbolic names for the CCSIDs that are supported by Text Search Engine. The 2-byte binary value must be specified in big-endian format.

Note that for searches on Ngram indexes, there is no choice of using a CCSID other than the one the index was built for. Specification of CCSID on a per search term base is ignored. Since supported CCSIDs for DBCS are combined code pages, queries on an Ngram index can contain sequences of SBCS characters.

LANG
Specifies the language identifier for the current search term. File IMOLANG.H, provided with Text Search Engine, defines symbolic names for the language identifiers that are supported by Text Search Engine. The 2-byte binary value must be specified in big-endian format.

SUBT
This flag is not for use with Ngram indexes. It requests that the search term is a subterm of the first search term to its left (within the same search argument), that is not a subterm.

Use this flag to characterize terms that have lower relevance for ranking than a search term they are semantically related to.

TERM
Specifies the search term to be compared to the textual content of the indexed documents. This search term must contain text characters in the coded character set specified in the CCSID item. It can also contain masking symbols defined with RCM or CSM items.

The search term is scanned to determine the complete words and the masked words. Scanning, the distinction of valid alphanumeric characters and word separators or punctuation marks, is based on the language specified with the LANG item.

Complete words within a search term are then normalized according to the rules established for text in the specific national language or via the index type.

If requested, expansions of words in the search terms are added.

Mask settings

For the positioning of mask settings, see “Search argument” on page 132. See “Example” on page 137 for examples of using masking symbols.

You can define several masking symbols (“wild-card” characters) for use in the current search term.

The datastream items are:

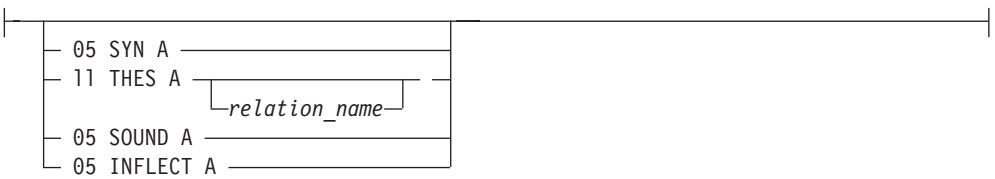
RCM
Defines the SBCS masking symbol for a single required character.

CSM
Defines the SBCS masking symbol for a sequence of optional characters or for a single optional word.

The masking symbols must be defined for each search term in which they are used. Each symbol must be a single character in the CCSID used for the search term. If both masking symbols (RCM and CSM) are specified in one search term, these symbols must be different.

For Ngram indexes:
- Only SBCS symbol characters, but no alphanumerical characters, are allowed as masking characters.
- Mask characters can only be specified adjacent to alphabetic characters. Masking for nonalphabetic characters is not supported.

Expansion qualifiers



For the positioning of expansion qualifiers, see “Search argument” on page 132 or “Free-text argument” on page 136.

Note that for Ngram indexes, only the THES qualifier is supported. Masking is not allowed for search terms that are to be expanded.

SYN
An option that requests to search also for the synonyms of the current search term. Text Search Engine uses the synonyms defined in the language dictionary on the server workstation.

THES
An option that requests to search also for thesaurus expansions of the current search term. Do not use an arbitrary phrase for a thesaurus search term. Use only thesaurus search terms that are likely to be found in the thesaurus dictionary, that is, single words or multiword terms like “national security”. The search engine looks for thesaurus terms either in the file defined by the global default keyword THESNAME or in the default file `imothes` for indexes of type XTYP_LINGUISTIC and XTYP_PRECISE, or in the default file `imonthes` for Ngram indexes.

If *relation_name* is specified, query expansion by thesaurus is done along branches of the named relation. If no value is specified, all branches are taken into account for query expansion.

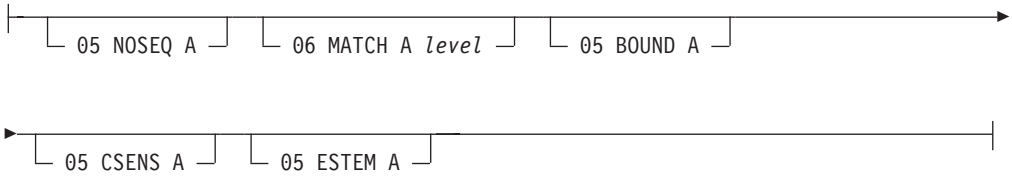
For an Ngram index, *relation_name* must be one of the following strings, where <n> is a number from 1 to 128:
SYNONYM
RELATED
RELATED<n>

Note that even on systems with DBCS CCSIDs, only the SBCS variant of the characters can be used for *relation_name*. Refer to “Relations” on page 11 for further information on thesaurus relations.

SOUND
An option that requests to search for terms that sound like the specified search term.

INFLECT
Useful only for indexes of type XTYP_PRECISE or XTYP_PRECISE with the additional feature XTADD_NORMALIZED. Requests expansion of input search terms using their lemma forms. For instance, search term `lost` is expanded to include `lose, losing, loses`.

Precision qualifiers



For the positioning of precision qualifiers, see “Search argument” on page 132.

The datastream items are:

NOSEQ
Requests that words in the current search term can occur in any sequence in a single sentence in the document text.

If not specified, words in the current search term must occur in exactly the same sequence in a single sentence in the document text.

NOSEQ cannot be specified for search terms in an EXCL text criterion or for searches on an Ngram index.

The following qualifiers are valid for Ngram indexes only.

MATCH
Specifies the degree of similarity requested for the search term. If you do not specify a MATCH level, then a search is made for an exact match. If you do specify a MATCH level, a search is made for a less-than-exact match. Valid values of *level* are hexadecimal X'01' to X'05', where X'05' is more exact than X'01'.

The matching level does not apply to the first three characters of a search term. That means that the first three characters of a search term must always match exactly.

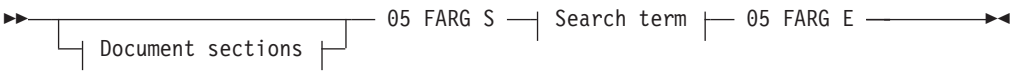
Note that usage of keyword MATCH excludes usage of masking symbols and vice versa.

BOUND
Requests search to respect word phrase boundaries. Valid only for Ngram indexes that were created for Korean CCSID.

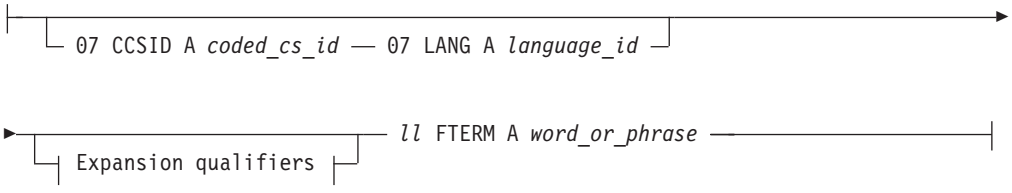
CSENS
Requests search to respect case, that is, the search is case-sensitive. This is valid only for Ngram indexes with the XTADD_CASE_ENABLED option.

ESTEM
Requests to search also for tokens having the same stem as TERM. If TERM is “compute”, for example, Text Search Engine searches also for “computing”, “computed”, “computer”, and so on. This is useful only if TERM is an English term.

Free-text argument



Search term:



Free-text search is not supported for Ngram indexes.

The datastream items are:

FARG
Delimits the free-text search argument. Only one free-text search argument per query and one search string per free-text argument is allowed. If a free-text search argument is preceded by a Boolean query, the result of such a combined Boolean and free-text query is a subset of the result of the Boolean part of the query. In other words, the Boolean and free-text part of a query are always (implicitly) combined by an AND operator.

CCSID
Specifies the SAA Coded Character Set Identifier for the current search string. File IMOLANG.H, provided with Text Search Engine, defines symbolic names for the CCSIDs that are supported by Text Search Engine. The 2-byte binary value must be specified in big-endian format.

LANG
Specifies the language identifier for the current search string. File IMOLANG.H, provided with Text Search Engine, defines symbolic names for the language identifiers that are supported by Text Search Engine. The 2-byte binary value must be specified in big-endian format.

Expansion qualifiers
The following keywords are supported:
SYN
THES
INFLECT

FTERM
Specifies the actual free-text query which can be single words, phrases or even whole sentences. If FTERM contains several words (for example, a phrase or a sentence), in contrast to a Boolean query these words need not occur in this sequence (*adjacent*) in a document.

Ranking is calculated using contributions from the individual words in FTERM and any lexical affinities that are formed. For further details see the description of the EhwRank function. The search term must contain text characters in the coded character set specified in the CCSID item.

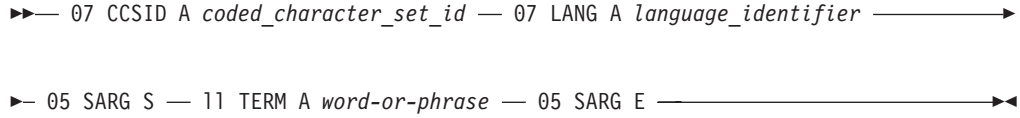
Note that masking of characters or words is not supported for search strings in a free-text argument.

The search term is scanned to determine the complete words. Scanning, the distinction of valid alphanumeric characters and word separators or punctuation marks, is based on the language specified with the LANG item.

More complex data stream items are “Document sections” on page 130 and “Expansion qualifiers” on page 134. See “Example” for an example of the datastream syntax for a simple free-text query.

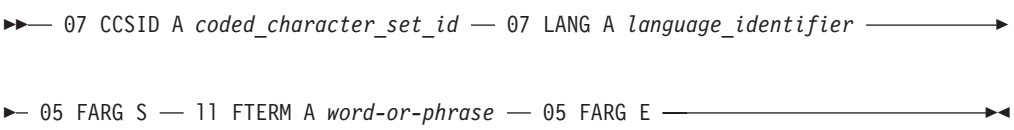
Example

Data stream for the simplest Boolean query



See “Boolean query” on page 129 for enhanced syntax.

Data stream for the simplest free-text query



See “Free-text argument” on page 136 for enhanced syntax.

An example of an EhwSearch function call

```
/*-------------------------------------------------------------------*/
/* issue an API function call to start a search                      */
/*-------------------------------------------------------------------*/
ulReturnCode =
EhwSearch (pSession,                /* In  -- session pointer      */
           ulIndexHandle,           /* In  -- index handle         */
           ulDataLength,            /* In  -- data stream length   */
           pDataStream,             /* In  -- query                */
           &ulResultHandle,         /* Out -- result handle        */
           &ulResultSize,           /* Out -- result size          */
           &ulDiagnosisInfo);       /* Out -- diagnosis info       */

if (ulReturnCode != RC_DONE)        /* check the API return code   */
{                                   /* handle the function error   */
}                                   /* endif API call failed       */

if (ulResultSize != 0L)             /* check the result size       */
{
                                    /* access the result list      */
}                                   /* endif result size not zero  */
/*-------------------------------------------------------------------*/
```

The area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would have a value of 128.

```
Symbolic notation              Hexadecimal representation

07 CCSID A CCSID_00819         0007 0073 C1 0333
07 LANG  A LANG_ENU            0007 0074 C1 177B
05 SARG  S                     0005 00C8 E2
20 TERM  A computer system     0014 00D3 C1 636F6D...
05 SARG  E                     0005 00C8 C5
05 AND   A                     0005 0082 C1
05 SARG  S                     0005 00C8 E2
06 RCM   A %                   0006 0168 C1 25
06 CSM   A *                   0006 015E C1 2A
05 NOSEQ A                     0006 00E5 C1
19 TERM  A A%me Computers      0013 00D3 C1 41256D...
06 RCM   A %                   0006 0168 C1 25
06 CSM   A *                   0006 015E C1 2A
05 NOSEQ A                     0006 00E5 C1 08
16 TERM  A %%rner Bro*         0010 00D3 C1 252572...
05 SARG  E                     0005 00C8 C5
```

EhwSelectResultView

This function forms a new result list from an existing one by selecting members from the list according to given criteria.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

result list handle
Identifies the search result list to base the selection on. It is the handle returned by EhwCreateResultView or EhwSelectResultView itself.

data stream length
The length, in bytes, of the data contained in the *selection criteria* parameter.

selection criteria
Specifies the criteria according to which a new result list is selected from the existing one. The criteria are specified in a data stream format (see “Data stream syntax” on page 140).

Output parameters

result list handle
Identifies the search result list created by EhwSelectResultView. It is a subset of the input result list.

result list size
The size of the result list.

diagnosis information
For a return code RC_DATASTREAM_SYNTAX_ERROR, the diagnosis information contains the offset of the erroneous data stream item.

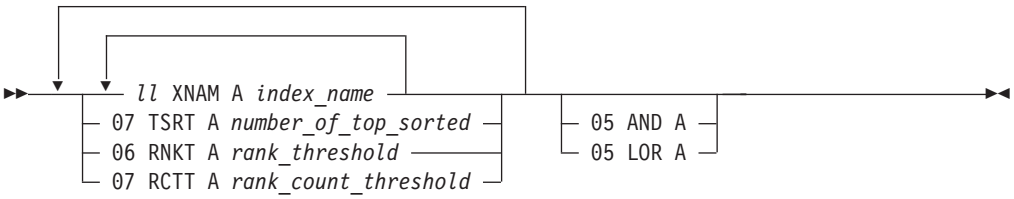
return code
The following return code values can be returned with this call:

RC_DONE
RC_EMPTY_LIST
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_REQUEST_IN_PROGRESS
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_NO_RANKING_DATA_AVAILABLE
RC_DATASTREAM_SYNTAX_ERROR
RC_NOT_ENOUGH_MEMORY
RC_INDEX_NOT_OPEN
RC_COMMUNICATION_PROBLEM
RC_MAX_NUMBER_OF_INDEXES

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Selection criteria



The data stream items are:

TSRT
Number of elements, counted from the beginning of the list, that are to be selected.

RNKT
A minimum relevance value of a document, where a relevance value is an unsigned char ranging from `0x00` to `0x64`.

RCTT
A minimum rank count of a document. The rank count is the total number of matches in the document that contributed to term ranking.

AND
All criteria must be met.

LOR
At least one of the criteria must be met.

XNAM
The name of an index that the document is associated with.

Usage

If several indexes are specified, these are always alternatives of each other, regardless of the AND or LOR operator specified. Each criterion can be specified at most once, and there must be at least one. If no AND or LOR operator is specified, then AND is assumed.

A call for EhwSelectResultView is possible only if the preceding search has been performed with the RANK option.

Selections based on rank count are not possible if the result list was produced for a query containing a free-text argument.

Selections based on rank value are possible only if EhwRank has run.

Example

This is an example of an EhwSelectResultView function call:

```
/*-------------------------------------------------------------------*/
/* Call API function EhwSelectResultView                             */
/*-------------------------------------------------------------------*/
   ulReturnCode =
            EhwSelectResultView (
                  pSession,         /* In  -- session pointer      */
```

```
                    ulResVwHandle,    /* In  -- result view handle   */
                    ulDataLength,     /* In  -- data stream length   */
                    pDataStream,      /* In  -- selection criteria   */
                    &pulResViewHandle,/* Out -- result view handle   */
                    &pulResViewSize,  /* Out -- size of result view  */
                    &ulDiagnosisInfo);/* Out -- diagnosis info.      */
                                      /* check the API return code   */
      if (ulReturnCode != RC_DONE)
      {
                                      /* handle the function error   */
      }                               /* endif API call failed       */


   /*-------------------------------------------------------------------*/
```

The area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would then have a value of 54.

```
Symbolic notation              Hexadecimal representation

12 XNAM  A INDEX07             000C 003C C1 494E4445583037
12 XNAM  A INDEX08             000C 003C C1 494E4445583038
12 XNAM  A INDEX09             000C 003C C1 494E4445583039
07 TSRT A  812                 0007 026C C1 032C
06 RNKT A  72                  0006 0276 C1 48
05 LOR  A                      0005 008C C1
```

EhwSetIndexFunctionStatus

This function changes the status of the Text Search Engine search (EhwSearch) scheduling (EhwScheduleDocument), indexing (EhwUpdateIndex), and index merge (EhwReorgIndex) functions. Possible actions to change the status are:

- Enable or disable a function
- Reset a function having an error condition.

This function can be used on an index that is opened and not suspended.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

index handle
Identifies the information-retrieval index whose function status information is to be changed. It is the handle returned by EhwOpenIndex when the index is opened.

function identifier
Identifies the Text Search Engine function whose status is required.

FCT_SEARCH_INDEX
Text Search Engine search function

FCT_SCHEDULE_DOCUMENTS
Schedule documents function

FCT_INDEX_DOCUMENTS
Index documents function

FCT_MERGE_INDEX
Reorganize index function

action id
Identifies the action to be applied to a Text Search Engine function

ACT_ENABLE
Enables a Text Search Engine function so that it can be used

ACT_DISABLE
Disables the use of a Text Search Engine function

ACT_RESET
Resets an error that occurred while a Text Search Engine function was running. After being reset, the function can be started again.

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

return code
The following return code values can be returned with this call:
RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_SERVER_CONNECTION_LOST
RC_REQUEST_IN_PROGRESS
RC_NOT_ENOUGH_MEMORY

RC_INDEX_SUSPENDED
RC_MAX_NUMBER_OF_INDEXES
RC_COMMUNICATION_PROBLEM
RC_UNEXPECTED_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Example

This is an example of an EhwSetIndexFunctionStatus call.

```
/*-------------------------------------------------------------*/
/* Call EhwSetIndexFunctionStatus to change the function status  */
/*-------------------------------------------------------------*/
ulReturnCode =
EhwSetIndexFunctionStatus(           /* Set status of a function */
               pSession,          /* In  -- session pointer      */
               ulIndexHdl,        /* In  -- index handle         */
               ulFunctionId,      /* In  -- function ID          */
               ulActionId,        /* In  -- action requested     */
               &ulDiagInfo);      /* Out -- diagnosis info       */

if (ulReturnCode  != RC_DONE)        /* check the API return code*/
{
                                     /* handle the function error*/
}                                    /* endif API call failed    */
/*-------------------------------------------------------------*/
```

After completion of the function call, the function that was stopped by the Text Search Engine server after an error occurred can be used again.

EhwSetIndexingRules

This function sets up the default rules for assigning format, language and CCSID parameters to a document when these parameters cannot be detected automatically during indexing. It also determines the default model and format to be used for indexes of additional type XTPROP_SECTIONS_ENABLED.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

index handle
Identifies the information-retrieval index for which the indexing rules apply. It is the handle returned by EhwOpenIndex when the index is opened.

data stream length
The length, in bytes, of the data contained in the *indexing rules* parameter.

indexing rules
Specifies the indexing rules to be applied by Text Search Engine. This parameter is supplied in a data stream format (see “Data stream syntax” on page 145).

Output parameters

diagnosis information
If the return code is RC_DATASTREAM_SYNTAX_ERROR the diagnosis information returned is the offset to the data stream item in error. If the offset is zero, an item is missing.

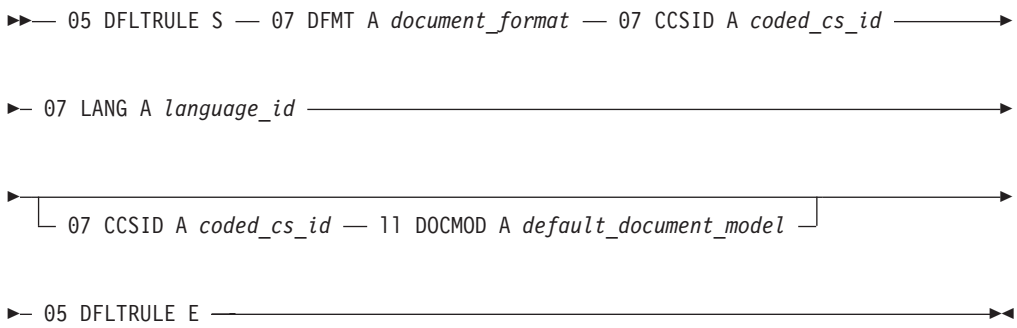
return code
The following return code values can be returned with this call:

RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_DATASTREAM_SYNTAX_ERROR
RC_MAX_INPUT_SIZE_EXCEEDED
RC_REQUEST_IN_PROGRESS
RC_INCORRECT_AUTHENTICATION (Text Search Engine for AIX only)
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_NOT_ENOUGH_MEMORY
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR
RC_DOCMOD_READ_PROBLEM
RC_UNKNOWN_DOCUMENT_MODEL_NAME

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Indexing rules



The data stream items are:

DFLTRULE
Delimits the default indexing rule.

DFMT
Specifies the default document format that is assumed by Text Search Engine when the format cannot be determined automatically. File IMOLSDEF.H, provided with Text Search Engine, defines symbolic names for the document formats that are supported by Text Search Engine. For indexes with property XTPROP_SECTIONS, the chosen format must be enabled for section recognition. For other indexes, the format does not have to be enabled for section recognition. The 2-byte binary value must be in big-endian format.

CCSID
Specifies the default SAA Coded Character Set Identifier that is applied by Text Search Engine when it cannot be determined automatically. File IMOLANG.H, provided with Text Search Engine, defines symbolic names for the CCSIDs that are supported by Text Search Engine. The 2-byte binary value must be in big-endian format.

For NGRAM indexes, the CCSID is ignored because it must not be altered once the index exists.

LANG
Specifies the default language that is assumed by Text Search Engine when it cannot be determined automatically. File IMOLANG.H, provided with Text Search Engine, defines symbolic names for the language identifiers that are supported by Text Search Engine. The 2-byte binary value must be in big-endian format.

CCSID
Specifies the SAA Coded Character Set Identifier for the following string input value. The 2-byte binary value must be in big-endian format.

DOCMOD
Specifies the default model to be used for section-enabled indexes. The model name given here must have been entered in the model definition file before creating the index. Input is invalid for all other indexes. Valid characters for the value of the DOCMOD parameter are in the range of [A-Z, a-z, 0-9].

Usage

When indexing a document, Text Search Engine needs to know:

- Which type of document it is; if it is an AmiPro document, for example, or a flat ASCII file.
- Which language is the document written in.
- Which Coded Character Set Identifier (CCSID) is used.

Most document formats are detected by Text Search Engine automatically, but a document does not always contain information about the language and CCSID. Text Search Engine allows you to specify default indexing rules that are applied whenever information about document format, language, or CCSID cannot be determined automatically.

When this call is issued to set default rules for a section-enabled index, ensure that the document format chosen as a default is enabled for section recognition. If it is set to any other value, the API call fails.

Use EhwGetIndexingRules to check which indexing rule is currently active.

Example

This is an example of an EhwSetIndexingRules function call:

```
/*-------------------------------------------------------------*/
/*  Call EhwSetIndexingRules to set the default indexing rule  */
/*-------------------------------------------------------------*/
ulReturnCode =
EhwSetIndexingRules(pSession,    /* In  -- session pointer      */
                    ulIndexHdl,  /* In  -- index handle         */
                    ulLength,    /* In  -- length of data stream*/
                    pchRule,     /* In  -- data stream          */
                    &ulDiagInfo);/* Out -- diagnosis info       */

if (ulReturnCode  != RC_DONE)    /* check the API return code   */
{
                                 /* handle the function error   */
}                                /* endif API call failed       */
```

EhwSort

This function sorts a result list according to the given criteria.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

result list handle
Identifies the search result list to be sorted. It is the handle returned by an EhwCreateResultView or EhwSelectResultView call.

data stream length
The length, in bytes, of the data contained in the *sort criteria* parameter.

sort criteria
Specifies the attributes and sort sequence according to which the result list is to be sorted. This sort method parameter has a data stream format (see “Data stream syntax”).

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

return code
The following return code values can be returned with this call:

RC_DONE
RC_EMPTY_LIST
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_REQUEST_IN_PROGRESS
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_NO_RANKING_DATA_AVAILABLE
RC_DATASTREAM_SYNTAX_ERROR
RC_NOT_ENOUGH_MEMORY
RC_COMMUNICATION_PROBLEM

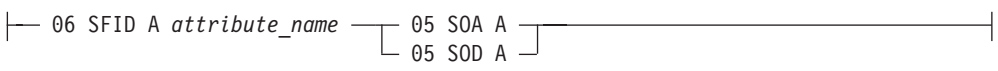
See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Sort criteria

Sort

Criterion:



The data stream items are:

SSP
Delimits a sort criterion.

SFID
Specifies the name of an attribute:

SFID_RVAL
Sort by 1-byte rank values.

SFID_DNAM
Sort by document name. Note that this is the document name Text Search Engine uses in its indexes. It can be different from names or titles a library assigns to this document.

SFID_XNAM
Alphanumerical sort by index name.

SFID_DSIZE
Sort by the number of occurrences of words in document (excluding stopwords).

Not available for the result of a query containing free-text arguments.

SFID_RCNT
Sort by the number of term matches in the document.

Not available for the result of a query containing free-text arguments.

SOA
Specifies ascending sort order.

SOD
Specifies descending sort order.

Usage

EhwSort supports multiple sort levels where each level has an independent sort direction. Each sort attribute can appear at most once in the sort data stream.

Example

This is an example of an EhwSort function call:

```
/*-------------------------------------------------------------------*/
/* Call API function EhwSort until end of data is indicated          */
/*-------------------------------------------------------------------*/
   ulReturnCode =
            EhwSort (pSession,      /* In  -- session pointer        */
                  ulResultHandle,   /* In  -- result view handle     */
                  &ulDataLength,    /* In  -- data stream length     */
                  &pDataStream,     /* In  -- sort method            */
                  &ulDiagnosisInfo);/* Out -- diagnosis info.        */
                                    /* check the API return code     */
   if (ulReturnCode != RC_DONE)
   {
                                    /* handle the function error     */
   }                                /* endif API call failed         */
```

```
    /*-----------------------------------------------------------------*/
    /* parse the result view data stream ...                           */
    /*-----------------------------------------------------------------*/
  /*-------------------------------------------------------------------*/
```

The area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would then have a value of 42.

```
Symbolic notation              Hexadecimal representation

05 SSP  S                      0005 0258 E2
06 SFID A SFID_RVAL            0006 0262 C1 01
05 SOD  A                      0005 0268 C1
05 SSP  E                      0005 019A C5
05 SSP  S                      0005 0258 E2
06 SFID A SFID_XNAM            0006 0262 C1 04
05 SOA  A                      0005 0267 C1
05 SSP  S                      0005 0258 E2
```

EhwStartSession

This function connects your application to a specified Text Search Engine server and starts an information-retrieval session. You must call this function before using any other Text Search Engine function except EhwListServers.

Input parameters

data stream length
The length, in bytes, of the data contained in the *session information* parameter.

session information
Identifies the Text Search Engine server. This parameter is supplied in a data stream format (see “Data stream syntax”).

Output parameters

session pointer
Identifies the information-retrieval session established. The session pointer is provided as an input parameter in subsequent function calls.

diagnosis information
There is no diagnosis information returned for this call.

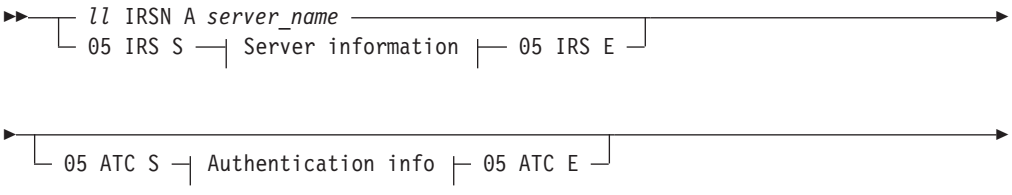
return code
The following return code values can be returned with this call:

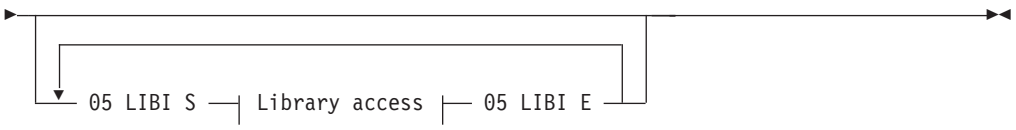
RC_DONE
RC_UNKNOWN_SERVER_NAME
RC_UNKNOWN_COMMUNICATION_TYPE
RC_UNKNOWN_SERVER_INFORMATION
RC_DATASTREAM_SYNTAX_ERROR
RC_INCORRECT_AUTHENTICATION
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_NOT_ENOUGH_MEMORY
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

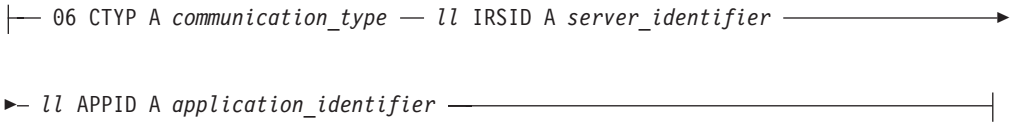
Data stream syntax

Session information

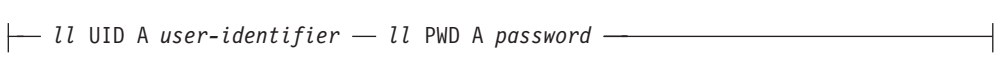




Server information:



Authentication info:



Library access:

The data stream items are:

IRSN
Specifies the symbolic name of the Text Search Engine server with which your application is to establish a session. It must be one of the server names returned by EhwListServers.

IRS
Delimits the information that specifies the requested information-retrieval session.

ATC
Delimits the authentication information. Authentication information is needed only when you start a service in which one of the following administration functions are used on an AIX operating system:
- EhwCreateIndex
- EhwDeleteIndex
- EhwReorgIndex
- EhwResumeIndex
- EhwSuspendIndex
- EhwUpdateIndex
- EhwSetIndexFunctionStatus
- EhwSetIndexInfo
- EhwClearIndex
- EhwClearScheduledDocuments

LIBI
Delimits the library access information.

CTYP
Specifies the type of the communication between client and Text Search Engine server.

The *communication type* is a one-byte code:

CTYP_PIPES
Windows NT only. Specifies that the client server communication works with named pipes.

CTYP_TCPIP
Specifies that the client server communication works with TCP/IP.

Additional communication protocols are available. They can be configured using the command line utilities `imocfgcl` for a client, and `imocfgsv` for a server. Use the datastream item IRSN to use one of these communication protocols.

IRSID
Specifies the identifier of the Text Search Engine server (LAN ID of the machine) with which your application is to establish a session.

APPID
Specifies the communication-specific application identifier. For communication type CTYP_PIPES, this is the name of the pipe; for CTYP_TCPIP this indicates the port number.

UID
When used with ATC, identifies the user for whom the information-retrieval session is to be started. *User-identifier* must be a user ID on the target machine, that is, a member of the Text Search Engine administrator group.

When used with LIBI, specifies the user ID for accessing the library system.

PWD
Specifies the password of the user identified by the UID item.

LIBNAM
Specifies the name of the library to be accessed.

Usage

Your application can start parallel sessions to several Text Search Engine servers. All API calls of a single session, starting with EhwStartSession and ending with EhwEndSession, must be issued sequentially. To ensure this, issue the calls of one session from a single thread of your application program.

Thread reentrancy is ensured only when one session per thread is used.

Example

This is an example of an EhwStartSession function call:

```
/*-------------------------------------------------------------------*/
/* Start a session, using API function EhwStartSession              */
/*-------------------------------------------------------------------*/
ulReturnCode =
EhwStartSession (ulDataLength,      /* In  -- data stream length   */
                 pDataStream,       /* In  -- session information  */
                 &pSession,         /* Out -- session pointer      */
                 &ulDiagnosisInfo); /* Out -- diagnosis info.      */

if (ulReturnCode != RC_DONE)        /* check the API return code   */
{
                                    /* handle the function error   */
}                                   /* endif API call failed       */
/*-------------------------------------------------------------------*/
```

The area pointed to by pDataStream could contain the following data stream, and ulDataLength would have a value of 13.

```
Symbolic notation              Hexadecimal representation

13 IRSN A SERVER02             000D 000B C1 5345525645523032
```

EhwSuspendIndex

This function suspends an information-retrieval index from any administrative task except for EhwResumeIndex. If the processing condition SUSP_EXCEPT_SEARCH is set, searching can continue.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

data stream length
The length, in bytes, of the data contained in the *index name* parameter.

index name
Specifies the information-retrieval index to be suspended. This parameter is supplied in a data stream format (see "Data stream syntax" on page 155).

processing condition
The following suspend conditions can be specified with this call:

SUSP_ALL
Specifies that you want to suspend the information-retrieval index, forcing a possibly running task that updates, reorganizes, or searches the current index, to be stopped immediately. Queries are not allowed after EhwSuspendIndex has run.

SUSP_EXCEPT_SEARCH
Specifies that you want to suspend the information-retrieval index, forcing a possibly running task that updates or reorganizes the current index, to be stopped immediately. Queries are still allowed.

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

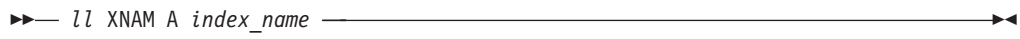
return code
The following return code values can be returned with this call:

RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_UNKNOWN_INDEX_NAME
RC_DATASTREAM_SYNTAX_ERROR
RC_UNKNOWN_CONDITION
RC_REQUEST_IN_PROGRESS
RC_INCORRECT_AUTHENTICATION (Text Search Engine for AIX only)
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_SERVER_IN_ERROR
RC_INDEX_NOT_ACCESSIBLE
RC_NOT_ENOUGH_MEMORY
RC_NO_ACTION_TAKEN
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Data stream syntax

Index name



The data stream item is:

XNAM
Specifies the name of the index to be suspended.

Usage

The EhwSuspendIndex function prevents the specified information-retrieval index from any modification until it is resumed with an EhwResumeIndex function call.

Example

This is an example of an EhwSuspendIndex function call:

```
/*-------------------------------------------------------------------*/
/* Call API function to suspend the information-retrieval index    */
/*-------------------------------------------------------------------*/
ulReturnCode =
EhwSuspendIndex (pSession,          /* In  -- session pointer      */
                 ulDataLength,      /* In  -- data stream length   */
                 pDataStream,       /* In  -- index name           */
                 &ulDiagnosisInfo); /* Out -- diagnosis info.      */

if (ulReturnCode != RC_DONE)        /* check the API return code   */
{
                                    /* handle the function error   */
}                                   /* endif API call failed       */
/*-------------------------------------------------------------------*/
```

The area pointed to by `pDataStream` could contain the following data stream, and `ulDataLength` would have a value of 12.

```
Symbolic notation             Hexadecimal representation

12 XNAM A INDEX07             000C 003C C1 494E4445583037
```

EhwUpdateIndex

This function starts a server task that updates an information-retrieval index.

Input parameters

session pointer
Identifies the information-retrieval session. It is the pointer returned by EhwStartSession when you start a session.

index handle
Identifies the information-retrieval index to be updated. It is the handle returned by EhwOpenIndex when the index is opened.

Output parameters

diagnosis information
There is no diagnosis information returned for this call.

return code
The following return code values can be returned with this call:

RC_DONE
RC_UNKNOWN_SESSION_POINTER
RC_INCORRECT_HANDLE
RC_REQUEST_IN_PROGRESS
RC_INCORRECT_AUTHENTICATION (Text Search Engine for AIX only)
RC_UNEXPECTED_ERROR
RC_SERVER_NOT_AVAILABLE
RC_SERVER_BUSY
RC_SERVER_CONNECTION_LOST
RC_CONFLICTING_TASK_RUNNING
RC_NOT_ENOUGH_MEMORY
RC_NO_ACTION_TAKEN
RC_FUNCTION_DISABLED
RC_FUNCTION_IN_ERROR
RC_COMMUNICATION_PROBLEM
RC_IO_PROBLEM
RC_WRITE_TO_DISK_ERROR
RC_MAX_NUMBER_OF_TASKS
RC_GTR_ERROR

See “Appendix A. The API return codes” on page 191 for more information about these codes.

Usage

The EhwUpdateIndex function starts the update of the specified Text Search Engine index. After checking whether the index can be updated and setting up the update process, control is returned to your application. The actual update process is performed asynchronously. That is, the return code RC_DONE signals that the update process has been started successfully. Your application is not notified when and if the update process has completed. Use EhwGetIndexFunctionStatus to check for the correct processing of administration tasks. If EhwGetIndexFunctionStatus returns an error code, refer to “Appendix B. Error codes returned by GetIndexingMsgs and GetIndexFunctionStatus” on page 207.

The update function processes the requests scheduled with EhwScheduleDocument calls since the last update of the index. Requests scheduled by other applications

for the same index are also processed. If the indexing queue contains several requests for the same document, only the latest request is processed.

Error messages and warnings related with EhwUpdate processing can be checked using the API call EhwGetIndexingMessages. For indexes of additional property XTPROP_SECTIONS_ENABLED the default settings for document model and document format must be compatible for EhwUpdate to run properly. See “EhwSetIndexingRules” on page 144 for details.

This is an example of an EhwUpdateIndex function call:

```
/*-------------------------------------------------------------------*/
/* Issue an API function call to start an update index task          */
/*-------------------------------------------------------------------*/
ulReturnCode =
EhwUpdateIndex (pSession,           /* In  -- session pointer        */
                ulIndexHandle,      /* In  -- index handle           */
                pulDiagnosisInfo);  /* Out -- diagnosis info         */

if (ulReturnCode != RC_DONE)        /* check the API return code     */
{
                                    /* distinguish the code types    */
   /*----------------------------------------------------------------*/
   /*                                  no error code:                */
   /* case RC_NO_ACTION_TAKEN:         empty scheduling queue        */
   /*                                  ...                           */
   /* case RC_CONFLICTING_TASK_RUNNING:                              */
   /*                                  active update | reorg. task   */
   /*                                  ...                           */
   /* case RC_FUNCTION_DISABLED:       update function is disabled   */
   /*                                  ...                           */
   /*                                  continue with processing      */
   /*----------------------------------------------------------------*/
   /*                                  function error:               */
   /* case RC_UNEXPECTED_ERROR:                                      */
   /* case RC_SERVER_BUSY:                                           */
   /* case RC_SERVER_CONNECTION_LOST:                                */
   /* case RC_COMMUNICATION_PROBLEM:                                 */
   /* case RC_NOT_ENOUGH_MEMORY:                                     */
   /* case RC_FUNCTION_IN_ERROR:                                     */
   /* case RC_IO_PROBLEM:                                            */
   /* case RC_WRITE_TO_DISK_ERROR:                                   */
   /*                                  handle the function error     */
   /*                                  ...                           */
   /*                                  stop processing / return      */
   /*----------------------------------------------------------------*/
   /* default:                         application error:            */
   /*                                  handle the internal error     */
   /*                                  ...                           */
   /*                                  stop processing / return      */
   /*----------------------------------------------------------------*/
}                                   /* endif API return code > 0     */
/*-------------------------------------------------------------------*/
```

UpdateIndex

Chapter 6. Connecting Text Search Engine to a library

This chapter explains the purpose of the Text Search Engine library service interfaces. It describes the types of library service requests issued by Text Search Engine and how you can implement these requests to connect Text Search Engine to your library system.

What library services provide

Using the Text Search Engine library service interfaces, you can connect Text Search Engine with the library system (or document management system) of your choice. The Text Search Engine library service interfaces are a set of defined requests that Text Search Engine issues to obtain services (such as access to documents) from the library system managing the documents indexed by Text Search Engine. This type of request interface is sometimes called a *user exit*.

This set of requests is designed to facilitate the mapping of the requests to the functional interfaces of any library system. The requests are defined as C function calls. All information exchange is done via parameters of these functions. Variable data, including entire documents, are passed in the form of data streams. (For general conventions on passing parameters and for an explanation of Text Search Engine data streams, see “Data exchange conventions” on page 1.)

To support a specific library system in your installation or for your application, you must program a full set of functions that map the Text Search Engine library service requests to the programming interfaces of that library system.

Text Search Engine requires library services on the client and on the server workstation so that, for example, documents can be accessed for indexing on the Text Search Engine server.

Functions

Here is a summary of the functions that library services need to provide:

- **Session control** functions are invoked by Text Search Engine to initiate or end a session with the library system. These functions are intended for allocating storage space and establishing an environment for subsequent function calls, and for releasing storage space and disconnecting from the library system.
- **Document access** functions are invoked by Text Search Engine to retrieve a document from the library system. These functions must also provide information about the document, such as the document format. The document content is retrieved in blocks of a size determined by Text Search Engine.
- **Document list** functions are invoked by Text Search Engine to obtain a list of the documents stored within a given document group, or a list of document groups within a given document group or in the entire library.
- **Attribute list** functions are invoked by Text Search Engine to retrieve attributes associated with a given document or document group.
- **Document index status** function is invoked by Text Search Engine to update the index status of documents in the library. This function is optional; it is only used when it exists.

Library service functions should respond to each call with a return code. In compliance with C programming conventions, the return code is the C function value. The possible return codes are defined by Text Search Engine for each function. Additional diagnosis information can also be returned, for example, to log the cause of a failure.

Detailed information about the parameters and use of each function is provided in “Chapter 7. Specifications for library services” on page 163.

Setting up the library connection

After you have programmed the library services required by Text Search Engine, link this set of C functions into a library services shared library.

These library services must be available on the Text Search Engine server and on each Text Search Engine client that can be connected to this server.

The library service requirements are slightly different for client and server:

- Some functions are not required on the server.
- Some functions request different levels of information on the client or server.

Details are specified for each individual function in “Chapter 7. Specifications for library services” on page 163.

It is recommended that you provide two different sets of library service functions, tailored to the requirements of the Text Search Engine client and server. This is especially useful when the differing requirements have performance implications. For example, the LIB_list_documents function can return a document name for each document. On the client, this name is displayed for the document and should be provided; on the server, this name is never used and should be omitted if obtaining it affects performance.

In the following, it is assumed that you provide two library services, one with client functions, the other with server functions. These DLLs must have 8-character names (with characters chosen from uppercase letters A-Z and digits 0-9).

These names must be specified as properties of the particular Text Search Engine index. Use the API function EhwCreateIndex or EhwSetIndexInfo to specify the library services names.

The functions are:

- *Client*:
  - LIB_init
  - LIB_access_doc
  - LIB_read_doc_content
  - LIB_close_doc
  - LIB_list_doc_groups
  - LIB_list_documents
  - LIB_get_doc_group_attr_values
  - LIB_get_doc_attr_values
  - LIB_end
- *Server*:
  - LIB_init
  - LIB_access_doc
  - LIB_read_doc_content
  - LIB_close_doc
  - LIB_list_doc_groups
  - LIB_list_documents
  - LIB_doc_index_status
  - LIB_end

For technical reasons on the AIX and z/OS platforms, shared libraries for the library services must contain a special function `exclsldr` (which can be a NOP function), and which must be defined as the main entry point for the shared library. Example:

```
#ifdef _AIX
#ifdef __cplusplus
extern "C" {void exclsldr(void);}
#endif // __cplusplus

void exclsldr(void)
{
   ; /* do nothing - required as main entry point */
}
#endif // _AIX
```

Only one set of library services can be associated with each Text Search Engine index. If you want to connect one Text Search Engine index with several library systems, you have to make the distinction (and the correct routing of requests) within the library service functions you provide.

Normally, you use one Text Search Engine index per library, so you can use one Text Search Engine server to support several library systems.

Text Search Engine provides the following library services for flat file access:

IMOLSCFS
: Client library service

IMOLSSFS
: Server library service

Finally, put the library services into the subdirectory specified by the corresponding environment variable (such as LIBPATH for z/OS, or PATH for Windows): both library services on the Text Search Engine server, the client library service on each Text Search Engine client.

On AIX systems, library services should be found in
`/usr/TextTools/lib`

On Sun Solaris systems, library services should be found in
`/opt/TextTools/lib`

Library services for z/OS offer access to documents on Hierarchical File System (HFS) and MVS datasets (PS, PO).

Chapter 7. Specifications for library services

This chapter describes each function of the Text Search Engine library service interfaces. It explains the input parameters that are provided and the output parameters that are returned. For each function, it gives a brief description of its usage within Text Search Engine.

All storage used to pass information from the library services to the calling Text Search Engine is allocated before the call. If not otherwise stated, the length of this storage is given using the same parameter used after the call to report how much of the storage has been used.

Where parameters are used for input and output, they are only listed as output parameters. For example, when Text Search Engine calls the function LIB_list_documents, it first gets the storage in which the function is to return a document list. The pointer to the storage space is passed as an input parameter but this parameter is not described. Instead, the document list that it points to is described as an output parameter.

For the exact number and sequence of parameters and their data types, refer to the file IMOLSPRO.H, which contains C prototype statements for each function. This file is part of the library services toolkit.

The set of return codes specifically for the library services are described in “Error handling concept and return codes” on page 185.

The library service functions are described in alphabetic order. Figure 23 shows the logical grouping of library service function calls.

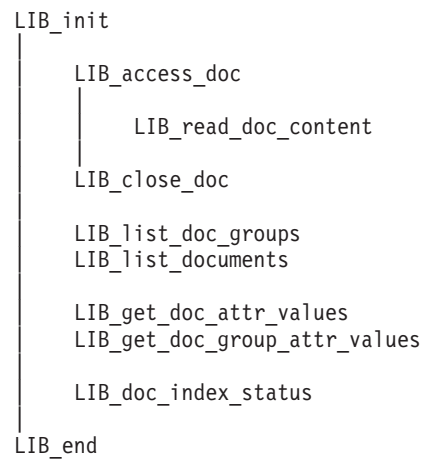


Figure 23. Library service functions and their logical relationships

LIB_access_doc

This function accesses and opens a document of the library system and returns the document information used by Text Search Engine. The *document information* output parameter is optional. If this parameter is not used, return a length of 0L for the length of document information value.

If any of the output values inside the datastream is omitted, return the correct length of the ouput datastream containing the other values.

If values for document information are returned, they override the default settings for the index the function was called for.

If format recognition of Text Search Engine is active, it may later override this default. To ensure that the document format setting returned by your own library service is used, switch off the format recognition of Text Search Engine (see the *Text Search Engine: Customization and Administration* manual).

Input parameters

anchor
Identifies the library session. It is the pointer returned by LIB_init.

length of document id
The length, in bytes, of the *document id* parameter.

document id
The identifier of the document to be accessed.

Output parameters

length of document information
The length, in bytes, of the *document information* parameter.

document information
Provides the document encoding information. This parameter is returned in data stream format (see “Data stream syntax” on page 165).

document handle
Identifies the accessed document and its status. It is used as input parameter for LIB_read_doc_content and LIB_close_doc.

diagnosis information
See “Error handling concept and return codes” on page 185 for information on the purpose of this parameter.

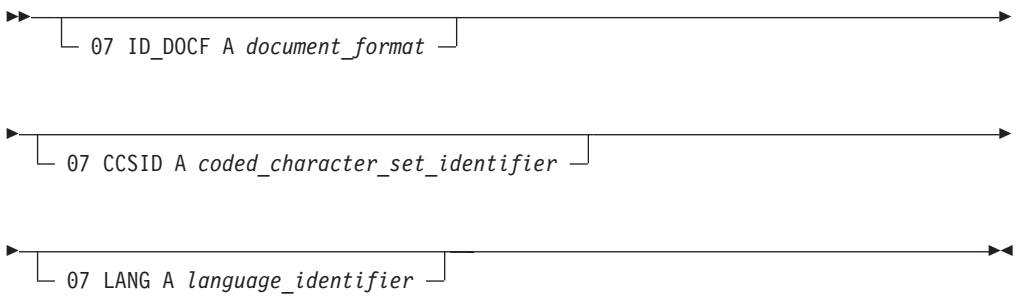
return code
Use one of the following return code values with this call:

RC_OK
RC_TERMINATION_ERROR
RC_DOCUMENT_CURR_NOT_ACCESSIBLE
RC_DOCUMENT_NOT_FOUND
RC_DOCUMENT_NOT_TO_INDEX
RC_DOCUMENT_IN_ERROR

See “Error handling concept and return codes” on page 185 for more information about these codes.

Data stream syntax

Document information



The data stream items are as follows:

ID_DOCF
Specifies the format of the document. *document format* is a two-byte binary code that must be specified in big-endian format. Symbolic names for all valid format codes are defined in file IMOLSDEF.H of the library services toolkit.

CCSID
Specifies the SAA Coded Character Set Identifier for text in the accessed document. This CCSID is assumed until control tags within the document itself specify a different CCSID or code page. File IMOLANG.H in the library services toolkit defines symbolic names for all CCSIDs supported by Text Search Engine. The two-byte binary value must be specified in big-endian format.

LANG
Specifies the language of the document. File IMOLANG.H in the library services toolkit defines symbolic names for all language identifiers that are supported by Text Search Engine. The 2-byte binary value must be specified in big-endian format.

If you specify a language for which no language dictionary is installed:

- The Text Search Engine indexing service cannot apply proper linguistic processing to this document. This is recorded as an indexing message that can be processed using EhwGetIndexingMsgs.

For documents in the Text Search Engine text format, the language is specified (and may change) within the document. For this document format, the language specified here is ignored.

Usage

This function is called to retrieve format and encoding information for a specific document. In addition, use the function to initialize subsequent reading (with LIB_read_doc_content calls) by setting the current position pointer to the beginning of the document content.

Text Search Engine accesses only one document at a time (within a single library session). It always calls LIB_close_doc before accessing another document.

LIB_close_doc

This function ends access to a document.

Input parameters

anchor
Identifies the library session. It is the pointer returned by LIB_init.

document handle
Identifies the document and its status. It is the handle returned by LIB_access_doc when the document was accessed. After completion of the call, this handle is considered obsolete.

Output parameters

diagnosis information
See “Error handling concept and return codes” on page 185 for information on the purpose of this parameter.

return code
Use one of the following return code values with this call:
RC_OK
RC_TERMINATION_ERROR
RC_DOCUMENT_IN_ERROR

See “Error handling concept and return codes” on page 185 for more information about these codes.

LIB_doc_index_status

This function allows the library to update the index status of documents. It is an optional function used on the Text Search Engine server: if there is no entry point defined in the library service on the server, it is not invoked.

Input parameters

anchor
Identifies the library session. It is the pointer returned by LIB_init.

length of document status list
The length, in bytes, of the *document status list* parameter.

document status list
Lists the documents whose index status has changed. This parameter is passed in data stream format as shown below.

check required
Allows the library service to do special checks when called during indexing of documents (LS_TRUE). When called while clearing the indexing queue or the index, this parameter is LS_FALSE. The symbolic names for this parameter are defined in file IMOLSDEF.H of the library services toolkit.

Output parameters

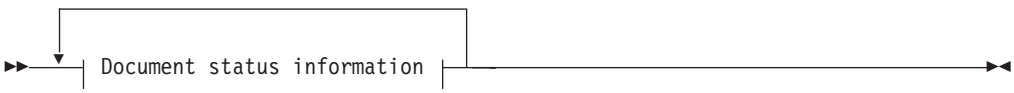
diagnosis information
See “Error handling concept and return codes” on page 185 for information on the purpose of this parameter.

return code
Use one of the following return code values with this call:
RC_OK
RC_TERMINATION_ERROR
RC_DATASTREAM_SYNTAX_ERROR

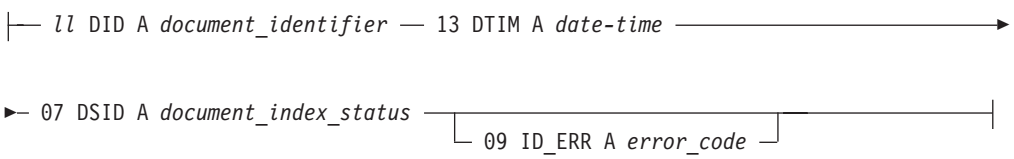
See “Error handling concept and return codes” on page 185 for more information about these codes.

Data stream syntax

Document status list



Document status information:



The data stream items are as follows:

DID
Specifies the identifier of a document whose index status has changed.

DTIM
Specifies the date and time when the document status changed. *date-time* is specified in the LS_DATIME format defined in file IMOLSDEF.H of the library services toolkit.

DSID
Specifies the changed index status.

Symbolic names for the document index status identifiers are defined in file IMOLSDEF.H of the library services toolkit.

ID_ERR
Specifies the reason why a document could not be indexed, or why a problem occurred when a document was indexed. *error code* is a 4-byte unsigned long. See “Appendix B. Error codes returned by GetIndexingMsgs and GetIndexFunctionStatus” on page 207.

Usage

This function is called to allow the library to update the index status of documents. It informs the library of the result of a requested indexing of documents, or clearing the index.

When called during the indexing of documents, the parameter *check required* is set to LS_TRUE, allowing the library to do checking, such as for consistency. When called during clear requests, it is set to LS_FALSE.

Any error when called while clearing the index causes the cleanup not to be done. When this function is called during indexing of documents, Text Search Engine performs the requested function against its index, regardless of the return code from this service. However, it tries to recover the problem by taking all documents in the call that caused the error and all documents in the next intended calls that have not been performed due to the error, and again calls the service within the next indexing of documents.

LIB_end

This function ends the library session established with the LIB_init call.

Input parameters

anchor
Identifies the library session. It is the pointer returned by LIB_init.

Output parameters

diagnosis information
See “Error handling concept and return codes” on page 185 for information on the purpose of this parameter.

return code
Use one of the following return code values with this call:
RC_OK
RC_TERMINATION_ERROR

See “Error handling concept and return codes” on page 185 for more information about these codes.

Usage

Use this function to release all storage areas allocated during the library session, and to end communication with the library system. After this call, the session anchor and any associated handles are considered obsolete by Text Search Engine.

LIB_get_doc_attr_values

This function returns attribute values for a given set of documents.

This function is not used for IBM Content Manager.

Input parameters

anchor
Identifies the library session. It is the pointer returned by LIB_init.

length of attribute request specification
The length, in bytes, of the *attribute request specification* parameter.

attribute request specification
Lists the attributes for which values are to be returned, and the documents for which these values are required. This parameter is passed in data stream format (see “Data stream syntax” on page 171).

length of buffer to receive attribute values list
The length, in bytes, of the buffer provided by Text Search Engine in which the function returns the requested information (see the *attribute values list* parameter). If the buffer is too small to receive all of the requested information, the information must be returned in blocks as described in “Usage” on page 172.

request type
Specifies the type of the request. It contains one of the values LS_FIRST, LS_NEXT, or LS_CANCEL. See “Usage” on page 172 for an explanation of request types. The symbolic names for request types are defined in file IMOLSDEF.H of the library services toolkit.

Output parameters

length of attribute values list
The length, in bytes, of the information returned in the *attribute values list* parameter.

attribute values list
Lists the requested attribute values for the requested documents. This parameter is returned in data stream format (see “Data stream syntax” on page 171).

attribute values handle
This is an output parameter for request types LS_FIRST and LS_NEXT when the return code is RC_CONTINUATION_MODE_ENTERED. The returned handle is used as an input parameter for request types LS_NEXT and LS_CANCEL to obtain the next block of information. See “Usage” on page 172 for a more detailed explanation.

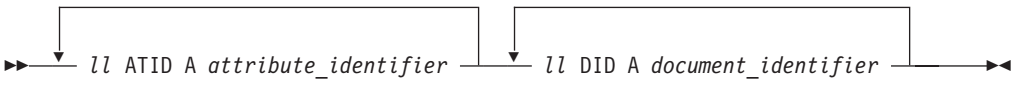
diagnosis information
See “Error handling concept and return codes” on page 185 for information on the purpose of this parameter.

return code
Use one of the following return code values with this call:
RC_OK
RC_TERMINATION_ERROR
RC_DATASTREAM_SYNTAX_ERROR
RC_NO_ATTRIBUTES_DEFINED

See “Error handling concept and return codes” on page 185 for more information about these codes.

Data stream syntax

Attribute request specification



The data stream items are as follows:

ATID
Specifies the identifier of an attribute whose value is requested. The following attribute identifiers are used by Text Search Engine:

NM
for the *document name* attribute

DT
for the *date-time* attribute

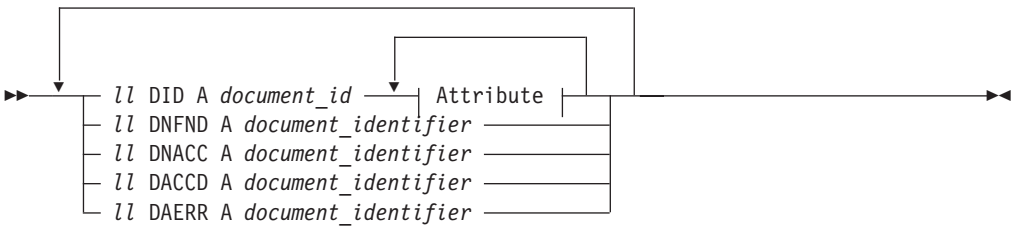
DS
for the *description* attribute

LO
for the *location* attribute.

Symbolic names for the attribute identifiers are defined in file IMOLSDEF.H of the library services toolkit.

DID
Specifies the identifier of a document for which attribute values are requested.

Attribute values list



Attribute:

The data stream items are as follows:

DID
Specifies the identifier of a document for which attribute values are returned.

ATID
Specifies the identifier of an attribute for which values were requested. The attributes must be returned in the same sequence as in the *attribute request specification*.

AVAL
Specifies the attribute value. If the attribute does not have a value (or is not maintained by your library system), no AVAL item is returned and the ATID item can also be omitted.

attribute value is returned as follows:
- For the DT attribute, in format LS_DATIME as defined in file IMOLSDEF.H of the library services toolkit. The length of the value is always 8 bytes.
- For all other attributes, as a zero-terminated character string in the system code page. These values are of variable length, with a maximum of 260 bytes.

DNFND
Specifies the identifier of a document that could not be found.

DNACC
Specifies the identifier of a document that is currently not accessible.

DACCD
Specifies the identifier of a document for which access is denied.

DAERR
Specifies the identifier of a document for which access failed.

The last four data stream items identify documents for which no attribute values can be returned. The documents should be listed in the same sequence as in the *attribute request specification*.

Usage

If none of the requested attributes is defined in your library system, use return code RC_NO_ATTRIBUTES_DEFINED (instead of returning a long list that contains no values).

The *request type* parameter is used as follows:
- The first request is always of type LS_FIRST.
- If the provided buffer is too small to contain all of the requested information, the first block of information is returned together with return code RC_CONTINUATION_MODE_ENTERED and a handle in the *attribute values handle* parameter.
- The next block of information is requested by calling the same function with request type LS_NEXT and providing the handle as input.
- This is repeated until the function returns the last block of information and return code RC_OK.
- If Text Search Engine encounters an error situation, or the remaining information is no longer required, the function is called with request type LS_CANCEL for cleanup processing.

LIB_get_doc_group_attr_values

This function returns attribute values for a given set of document groups.

Input parameters

anchor
Identifies the library session. It is the pointer returned by LIB_init.

length of attribute request specification
The length, in bytes, of the *attribute request specification* parameter.

attribute request specification
Lists the attributes for which values are to be returned, and the document groups for which these values are required. This parameter is passed in data stream format (see “Data stream syntax” on page 174).

length of buffer to receive attribute values list
The length, in bytes, of the buffer provided by Text Search Engine in which the function returns the requested information (see the *attribute values list* parameter). If the buffer is too small to receive all of the requested information, the information must be returned in blocks as described in “Usage” on page 175.

request type
Specifies the type of the request. It contains one of the values LS_FIRST, LS_NEXT, or LS_CANCEL. See “Usage” on page 175 for an explanation of request types. The symbolic names for request types are defined in file IMOLSDEF.H of the library services toolkit.

Output parameters

length of attribute values list
The length, in bytes, of the information returned in the *attribute values list* parameter.

attribute values list
Lists the requested attribute values for the requested document groups. This parameter is returned in data stream format (see “Data stream syntax” on page 174).

attribute values handle
This is an output parameter for request types LS_FIRST and LS_NEXT when the return code is RC_CONTINUATION_MODE_ENTERED. The returned handle is used as an input parameter for request types LS_NEXT and LS_CANCEL to obtain the next block of information. See “Usage” on page 175 for a more detailed explanation.

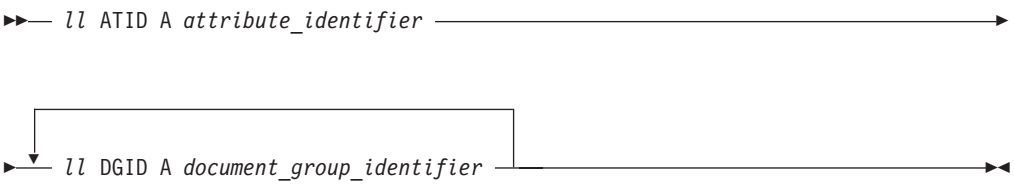
diagnosis information
See “Error handling concept and return codes” on page 185 for information on the purpose of this parameter.

return code
Use one of the following return code values with this call:
RC_OK
RC_TERMINATION_ERROR
RC_DATASTREAM_SYNTAX_ERROR
RC_NO_ATTRIBUTES_DEFINED

See “Error handling concept and return codes” on page 185 for more information about these codes.

Data stream syntax

Attribute request specification



The data stream items are as follows:

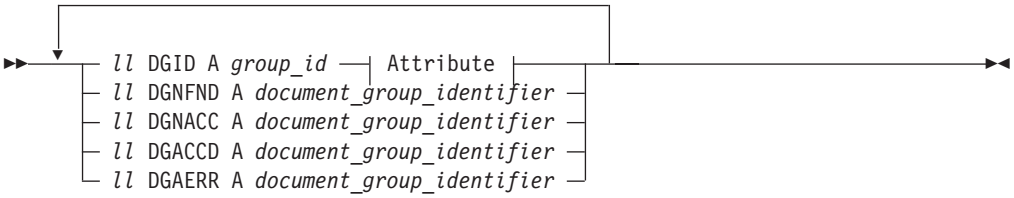
ATID
Specifies the identifier of an attribute whose value is requested. The following attribute identifier is used by Text Search Engine:

NM
for the *group name* attribute

Symbolic names for the attribute identifiers are defined in file IMOLSDEF.H of the library services toolkit.

DGID
Specifies the identifier of a document group for which attribute values are requested.

Attribute values list



Attribute:

The data stream items are as follows:

DGID
Specifies the identifier of a document group for which attribute values are returned.

ATID
Specifies the identifier of an attribute for which values were requested.

AVAL
Specifies the attribute value. If the attribute does not have a value (or is not maintained by your library system), no AVAL item is returned and the ATID item can also be omitted.

The *attribute value* for the NM attribute, the only attribute requested by Text Search Engine, is returned as a zero-terminated character string in the system code page. The value is of variable length, with a maximum of 260 bytes.

DGNFND
Specifies the identifier of a document group that could not be found.

DGNACC
Specifies the identifier of a document group that is currently not accessible.

DGACCD
Specifies the identifier of a document group for which access is denied.

DGAERR
Specifies the identifier of a document group for which access failed.

The last four data stream items identify document groups for which no attribute values can be returned. The document groups should be listed in the same sequence as in the *attribute request specification*.

Usage

If none of the requested attributes is defined in your library system, use return code RC_NO_ATTRIBUTES_DEFINED (instead of returning a long list that contains no values).

The *request type* parameter is used as follows:

- The first request is always of type LS_FIRST.
- If the provided buffer is too small to contain all of the requested information, the first block of information is returned together with return code RC_CONTINUATION_MODE_ENTERED and a handle in the *attribute values handle* parameter.
- The next block of information is requested by calling the same function with request type LS_NEXT and providing the handle as input.
- This is repeated until the function returns the last block of information and return code RC_OK.
- If Text Search Engine encounters an error situation, or the remaining information is no longer required, the function is called with request type LS_CANCEL for cleanup processing.

LIB_init

This function starts a library session. It is called by Text Search Engine before any other library service function.

Input parameters

data stream length
The length, in bytes, of the data contained in the *session information* parameter.

session information
Identifies the index name and (optionally the library) for which this session should be established. This parameter is supplied in a data stream format (see "Data stream syntax").

Output parameters

anchor
Identifies the library session established. This pointer is provided as an input parameter to all subsequent function calls, so data needed by them can be passed using this pointer. You can allocate storage for your own data, then "anchor" it in this parameter and use its data for subsequent calls. Remember to release this storage in the LIB_end call.

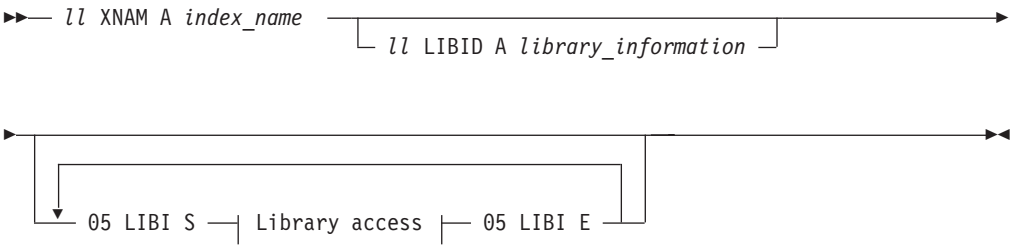
diagnosis information
See "Error handling concept and return codes" on page 185 for information on the purpose of this parameter.

return code
Use one of the following return code values with this call:
RC_OK
RC_TERMINATION_ERROR
RC_DATASTREAM_SYNTAX_ERROR

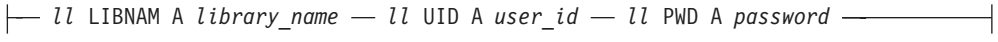
See "Error handling concept and return codes" on page 185 for more information about these codes.

Data stream syntax

Session information



Library access:



The data stream items are as follows:

XNAM
Specifies the current Text Search Engine index. You use it to do index-specific processing in your library services.

LIBID
Specifies the library information per index. It is the item specified with the EhwCreateIndex API call.

LIBI
Delimits the library access information.

LIBNAM
Specifies the name of the library to be accessed.

UID
Specifies the user ID for accessing the library system.

PWD
Specifies the password of the user identified by the UID item.

Usage

The *anchor* is a pointer to the environment established for communication with the library system. With only the anchor as an input parameter, the LIB_end function needs to be able to identify and free all storage allocated for the current session, and to end communication with the library system.

LIB_list_doc_groups

This function lists all document groups within a given document group or within the entire library.

Input parameters

anchor
Identifies the library session. It is the pointer returned by LIB_init.

length of document group id
The length, in bytes, of the *document group id* parameter.

document group id
The identifier of the document group for which a list of embedded document groups is requested. To request a list of document groups for the entire library, this parameter contains a special identifier defined with the symbolic name `LIBRARY` in file IMOLSDEF.H of the library services toolkit.

length of buffer to receive document group list
The length, in bytes, of the buffer provided by Text Search Engine in which the function returns the requested information (see the *document group list* parameter). If the buffer is too small to receive all of the requested information, the information must be returned in blocks as described in “Usage” on page 179.

request type
Specifies the type of the request. It contains one of the values LS_FIRST, LS_NEXT, or LS_CANCEL. See “Usage” on page 179 for an explanation of request types. The symbolic names for request types are defined in file IMOLSDEF.H of the library services toolkit.

Output parameters

length of document group list
The length, in bytes, of the information returned in the *document group list* parameter.

document group list
Lists the identifiers (and, optionally, the names) of document groups embedded in the document group or library specified in the *document group id* parameter. This parameter is returned in data stream format (see “Data stream syntax” on page 179).

doc group list handle
An output parameter for request types LS_FIRST and LS_NEXT when the return code is RC_CONTINUATION_MODE_ENTERED. The returned handle is used as an input parameter for request types LS_NEXT and LS_CANCEL to obtain the next block of information. See “Usage” on page 179 for a more detailed explanation.

diagnosis information
See “Error handling concept and return codes” on page 185 for information on the purpose of this parameter.

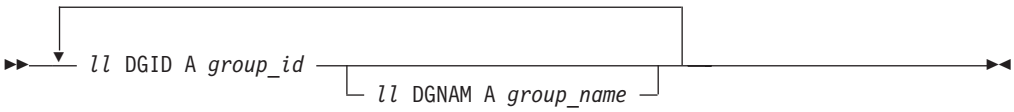
return code
Use one of the following return code values with this call:
RC_OK
RC_CONTINUATION_MODE_ENTERED
RC_TERMINATION_ERROR

RC_EMPTY_LIST
RC_DOCUMENT_GROUP_NOT_FOUND

See “Error handling concept and return codes” on page 185 for more information about these codes.

Data stream syntax

Document group list



The data stream items are as follows:

DGID
Specifies a document group identifier

DGNAM
Specifies the document group name. A document group name should be returned when the function is called from the Text Search Engine client.

Usage

This function supports the concept of document groups within library systems. Document groups can be implemented under various names (folders, workgroups, subdirectories) in different library systems. Text Search Engine is not concerned with the library-specific implementation of the group concept.

If your library system has a hierarchical group concept (allowing for groups within groups), implement the LIB_list_doc_groups function to handle only one nesting level at a time. That is, it lists only the document groups on the next level below the given group; if called for the entire library, the function lists only the highest-level groups in the library. Text Search Engine calls the function repeatedly to list the lower-level groups. Use the return code RC_EMPTY_LIST to indicate that there are no further groups embedded in the given group.

If your library does not support a group concept, implement the LIB_list_doc_groups function to return the RC_EMPTY_LIST return code when called for the entire library.

The *request type* parameter is used as follows:

- The first request is always of type LS_FIRST.
- If the provided buffer is too small to contain all of the requested information, the first block of information is returned together with return code RC_CONTINUATION_MODE_ENTERED and a handle in the *doc group list handle* parameter.
- The next block of information is requested by calling the same function with request type LS_NEXT and providing the handle as input.
- This is repeated until the function returns the last block of information and return code RC_OK.
- If Text Search Engine encounters an error situation, or the remaining information is no longer required, the function is called with request type LS_CANCEL.

Text Search Engine usually calls LIB_list_doc_groups for one document group, then, without retrieving all blocks of information, calls the function for a second document group, then requests the next block of information for the first document group, and so on. When programming this library service function, make sure that you support this calling sequence; use the *doc group list handle* to maintain the status of each pending function request.

Together with LIB_list_documents, this function is used to list all documents that belong to a specified document group.

LIB_list_documents

This function lists all documents within a given document group. The list can be restricted by a condition specified in the request, and the function can return some additional information for each document in the list.

Input parameters

anchor
Identifies the library session. It is the pointer returned by LIB_init.

length of document group specification
The length, in bytes, of the *document group specification* parameter.

document group specification
Contains the identifier of the document group for which documents are to be listed and, optionally, conditions for the document list. This parameter is passed in data stream format (see “Data stream syntax” on page 182).

length of buffer to receive document list
The length, in bytes, of the buffer provided by Text Search Engine in which the function returns the requested information (see the *document list* parameter). If the buffer is too small to receive all of the requested information, the information must be returned in blocks as described in “Usage” on page 182.

request type
Specifies the type of the request. It contains one of the values LS_FIRST, LS_NEXT, or LS_CANCEL. See “Usage” on page 182 for an explanation of request types. The symbolic names for request types are defined in file IMOLSDEF.H of the library services toolkit.

Output parameters

length of document list
The length, in bytes, of the information returned in the *document list* parameter.

document list
Lists the identifiers (and, optionally, some attributes) of the documents in the given document group. This parameter is returned in data stream format (see “Data stream syntax” on page 182).

document list handle
This is an output parameter for request types LS_FIRST and LS_NEXT when the return code is RC_CONTINUATION_MODE_ENTERED. The returned handle is used as an input parameter for request types LS_NEXT and LS_CANCEL to obtain the next block of information. See “Usage” on page 182 for a more detailed explanation.

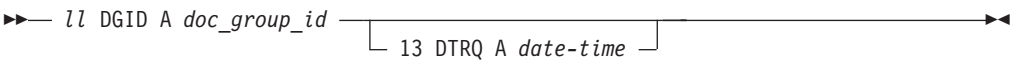
diagnosis information
See “Error handling concept and return codes” on page 185 for information on the purpose of this parameter.

return code
Use one of the following return code values with this call:
RC_OK
RC_CONTINUATION_MODE_ENTERED
RC_TERMINATION_ERROR
RC_EMPTY_LIST
RC_DATASTREAM_SYNTAX_ERROR
RC_DOCUMENT_GROUP_NOT_FOUND

See “Error handling concept and return codes” on page 185 for more information about these codes.

Data stream syntax

Document group specification



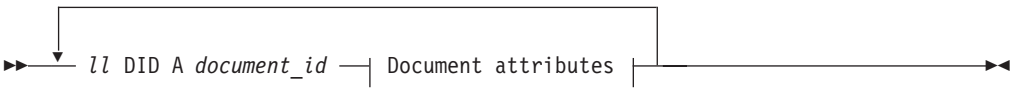
The data stream items are as follows:

DGID
Specifies the identifier of the document group for which documents are to be listed.

DTRQ
Specifies a date-time condition for documents to be listed. It restricts the list to documents added, changed, or deleted since the specified date and time. See “Usage” for further information.

date-time is specified in the LS_DATIME format defined in file IMOLSDEF.H of the library services toolkit.

Document list



Document attributes:

The data stream items are as follows:

DID
Specifies a document identifier.

DNAM
Specifies the document name. A document name should be returned when the function is called from the Text Search Engine client.

DTLM
Specifies the date and time when the document was last modified or when the document was created. *date-time* is specified in the LS_DATIME format defined in file IMOLSDEF.H of the library services toolkit.

Usage

If your library system has a hierarchical group concept (allowing for groups within groups), implement the LIB_list_documents function to list only the documents directly contained in the given group, not documents in subgroups of the given

group. Text Search Engine uses the LIB_list_doc_groups function repeatedly to list the lower-level groups, then the LIB_list_documents function to list the documents in these subgroups.

The *request type* parameter is used as follows:

- The first request is always of type LS_FIRST.
- If the provided buffer is too small to contain all of the requested information, the first block of information is returned together with return code RC_CONTINUATION_MODE_ENTERED and a handle in the *document list handle* parameter.
- The next block of information is requested by calling the same function with request type LS_NEXT and providing the handle as input.
- This is repeated until the function returns the last block of information and return code RC_OK.
- If Text Search Engine encounters an error situation, or the remaining information is no longer required, the function is called with request type LS_CANCEL.

Text Search Engine usually calls LIB_list_documents for one document group, then, without retrieving all blocks of information, calls the function for a second document group, then requests the next block of information for the first document group, and so forth. When programming this library service function, make sure that you support this calling sequence; use the *document list handle* to maintain the status of each pending function request.

On the Text Search Engine server, this function is used to list all documents that belong to a specified document group, and have been added or modified since the date and time specified (with the DTRQ item).

If your library system cannot handle a date-time condition (for example, because it does not maintain modification dates for documents), the function can return all documents that belong to the specified group.

On the Text Search Engine client, the function is used to display the documents in a given document group. The user can then select documents for indexing.

LIB_read_doc_content

This function reads a specified amount of the document content.

Input parameters

anchor
Identifies the library session. It is the pointer returned by LIB_init.

document handle
Identifies the document and its status. It is the handle returned by LIB_access_doc when the document was accessed.

bytes to be skipped
Number of bytes to be skipped before actually reading the document content, starting from the current position. Increase the current position pointer by the number of skipped bytes.

bytes to be read
Number of bytes to be read starting from the current position. After reading, increase the current position pointer by the number of bytes requested.

Output parameters

length of document text
The length, in bytes, of the *document text* actually returned. For return code RC_OK, this must be equal to the number of bytes requested (parameter *bytes to be read*). For return code RC_END_OF_FILE, this number may be lower than or equal to the number of bytes requested.

document text
Contains the requested part of the document content.

diagnosis information
See “Error handling concept and return codes” on page 185 for information on the purpose of this parameter.

return code
Use one of the following return code values with this call:
RC_OK
RC_TERMINATION_ERROR
RC_DOCUMENT_CURR_NOT_ACCESSIBLE
RC_END_OF_FILE
RC_DOCUMENT_IN_ERROR

See “Error handling concept and return codes” on page 185 for more information about these codes.

Usage

This function is called to read parts, or all of, the document content. On the Text Search Engine server, the function is required to read documents for indexing.

When browsing compound documents (that is, documents that contain images, graphics, or other pieces of information that are not displayed by the Text Search Engine browser), Text Search Engine uses the parameter *bytes to be skipped* to improve performance by skipping nontext information in the document content.

Error handling concept and return codes

Text Search Engine defines return codes for use with your library service functions. With these return codes, you can accurately record problems with document access or document handling, restrictions of your library system, and even errors in the function input provided by Text Search Engine. Some return codes, such as RC_OK or RC_END_OF_FILE, signal processing conditions to Text Search Engine. Only one return code, RC_TERMINATION_ERROR, is used to record an unrecoverable error situation.

With each call, you can return up to 16 bytes of additional information in the *diagnosis information* parameter. If supplied together with return code RC_OK, RC_CONTINUATION_MODE_ENTERED, RC_EMPTY_LIST, or RC_END_OF_FILE, Text Search Engine ignores the diagnosis information. If supplied together with any other return code value, Text Search Engine logs the diagnosis information.

It is recommended that you use this feature only as a complementary service and implement your own error logging facilities within the library services.

The following return codes are defined for library service functions:

RC_CONTINUATION_MODE_ENTERED
Indicates that the buffer is too small to return all of the requested information. Further calls are required to obtain the remaining information.

RC_DATASTREAM_SYNTAX_ERROR
Indicates an error in the input data stream provided by Text Search Engine. This return code causes Text Search Engine to end the processing.

RC_DOCUMENT_GROUP_NOT_FOUND
Indicates that the specified document group is not found. Text Search Engine skips processing of the group.

RC_DOCUMENT_IN_ERROR
Indicates that an error occurred or is detected while processing the requested document. This causes Text Search Engine to end the processing of the document. If the document is accessed for indexing, this is recorded as a document error and the indexing task continues to process the remaining documents scheduled for indexing.

RC_DOCUMENT_CURR_NOT_ACCESSIBLE
Indicates that the requested document is currently not accessible. The document may, for example, be accessed exclusively by another task or user. This causes Text Search Engine to:
- Show an error message if it was to display the document
- Defer the indexing of the document if it was to read the document for indexing.

RC_DOCUMENT_NOT_FOUND
Indicates that the requested document is not found. This situation can occur when a document was deleted but not yet removed from the index, or when a document was deleted after being scheduled for indexing. It causes Text Search Engine to:
- Show an error message if it was to display the document
- Skip the indexing of the document if it was to read the document for indexing.

RC_DOCUMENT_NOT_TO_INDEX
Indicates that the requested document is not to be indexed. This situation can occur when the document is scheduled for indexing, but is actually not suited for indexing (for example, because it does not contain text). This return code causes Text Search Engine to skip the indexing of the document.

RC_EMPTY_LIST
Indicates that the requested list of documents, document groups, or attributes is empty. It serves as an informational processing condition.

RC_END_OF_FILE
Indicates that the end of the document content was reached.

RC_NO_ATTRIBUTES_DEFINED
Indicates that none of the specified attributes are defined in your library system.

RC_OK
Indicates that the function completed successfully.

RC_TERMINATION_ERROR
Indicates that an unrecoverable error situation was detected that requires ending the calling Text Search Engine task. This return code should be used with care and only when there is no other possibility than to terminate the processing of Text Search Engine.

Chapter 8. Using the Text Search Engine text format

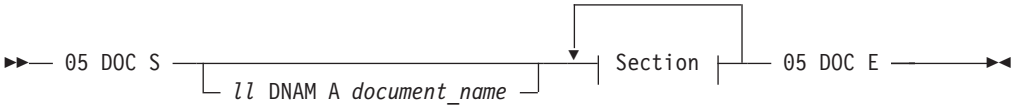
The Text Search Engine text format is a document format that consists of plain text and a few controls required for linguistic processing of the text. Its purpose is to provide a simple data interface for text-processing systems (such as editors), library systems, or customer applications that intend to have documents indexed by Text Search Engine.

Documents in this format can be processed by the Text Search Engine document-indexing functions. The Text Search Engine text format is a data stream format. See “Basics about the data streams” on page 2.

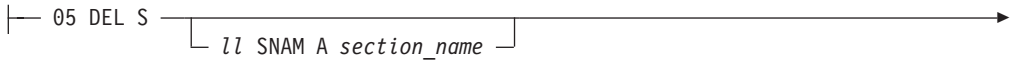
Data stream syntax

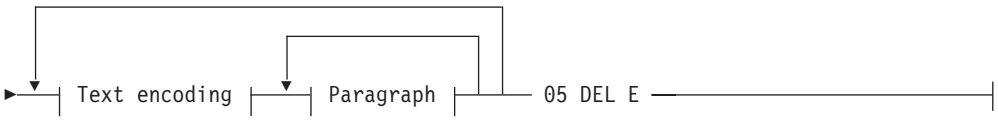
The following diagram defines a *document* in Text Search Engine text format:

Document

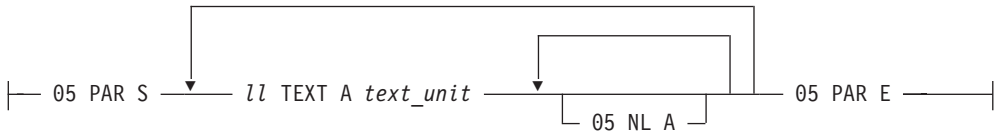


Section:

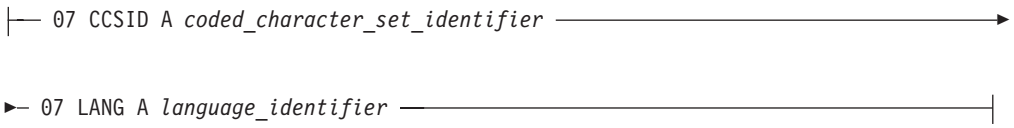




Paragraph:



Text encoding:



The data stream items are as follows:

DOC
Indicates the start and end of a document.

DNAM
Specifies a document name. This specification is optional and is not used by Text Search Engine.

DEL
Indicates the start and end of a document element. The only type of document element currently supported is a *text section*.

SNAM
Specifies the name of a text section. Section names should conform to the rules for Text Search Engine names given in “Passing character data” on page 3. Currently, Text Search Engine supports only one text section with a default name. The specification of a section name is ignored and can be omitted.

PAR
Indicates the start and end of a text paragraph within the current section. Paragraphs can be used in Text Search Engine search requests as a proximity measure.

TEXT
Specifies one text portion within the current paragraph. Usually, *text unit* contains one line of text, and the TEXT item is followed by an NL item; but text lines can also be split into several parts, each part specified in its own TEXT item.

The text should use the coded character set and language associated with the current paragraph.

NL
Indicates the start of a new line in the current paragraph. Lines are not used as a proximity measure in Text Search Engine search requests; Text Search Engine uses *sentences* as the more natural proximity measure. NL items can, therefore, be omitted entirely.

During indexing, NL items trigger the Text Search Engine dehyphenation process: the removal of syllable hyphens from line ends and reunification of the fragments of hyphenated words.

CCSID
Specifies the SAA Coded Character Set Identifier for text in subsequent paragraphs, which remains valid until a paragraph is preceded by a new CCSID item. You can use one of the following CCSIDs:

`CCSID_00500`
for text in the Latin-1 EBCDIC code page 500.

`CCSID_0850`
for text in the Latin-1 ASCII code page 850.

`CCSID_00819`
for text in the ASCII code page 819 as defined by ISO standard 8859-1.

These symbolic names for CCSIDs are defined in the file IMOLANG.H of the library services toolkit provided with Text Search Engine. The 2-byte binary value must be specified in big-endian format.

LANG
Specifies the language identifier for text in subsequent paragraphs, valid until a paragraph is preceded by a new LANG item. File IMOLANG.H in the library

services toolkit defines symbolic names for all language identifiers supported by Text Search Engine. The 2-byte binary value must be specified in big-endian format.

Symbolic names for the item identifiers of the above data stream items are defined in the file IMOLSDEF.H provided with the Text Search Engine toolkits.

Appendix A. The API return codes

This appendix lists the codes that are returned by the Text Search Engine API in response to a function call. They are listed in alphabetic order. The meaning of each code is explained and the recommended action for it is given. It also gives general information on error handling.

All Text Search Engine API calls return a numeric return code as the C function value. Throughout this book, symbolic names are used, such as RC_INCORRECT_HANDLE. The actual return codes are defined in file IMOAPIC.H of the API toolkit, provided with Text Search Engine. In this file, the return codes are listed by type alphabetically and by value in a separate table. The descriptions of each RC also is in alphabetic order. See “Setting up your application” on page 34 for recommendations on using the toolkit definitions in your application program.

API error handling

The Text Search Engine API intercepts error situations and reports error conditions with a return code. Additional diagnosis information can also be returned.

Applications that call Text Search Engine API functions should always check the return code before trying to process any other output parameters. The return codes possible with each call are listed with the parameters of each call in “Chapter 5. Calling the API functions” on page 37. A detailed explanation of each return code is given in this chapter.

The diagnosis information is provided to help you find the cause of an error. It is a 4-byte binary value. It should not be processed by your application, but only used for problem determination. To help you do so, you can print, display, or dump the diagnosis information.

In some cases, incorrect input such as an obsolete *session pointer* can cause an abnormal end condition in the API services that cannot be intercepted by Text Search Engine.

Types of return codes

The API return codes can be grouped as follows:

- **No error**

  These codes are returned when a function call has completed successfully. If there is no other action to perform and no supplementary information available, an RC_DONE code is returned. When only part of the requested information is returned because the buffers cannot store all of it, an RC_CONTINUATION_MODE_ENTERED code is returned.

  Other codes in this category include information that your application may want to pass on to users or to an administrator.

  Your application program should check for each return code of this type and take appropriate action.

- **User error**

  Return codes of this type indicate that there is an error in an input parameter of the function call. These are usually due to a programming error in your

application. Thus, your application should only record return codes of this type and take no further action. You should, of course, modify the program that caused the error.

- **Internal program error**

  This code indicates an error in one of the Text Search Engine programs. If you cannot continue using the product, you should contact your IBM representative.

- **Resource constraints**

  These codes are returned when there is a problem with storage space or other resources. In some cases, your application can free local resources (such as search results that are no longer needed).

- **Other errors**

  These can occur for several reasons, for example, when the Text Search Engine server is not available. Your application should only record return codes of this type. Then, an administrator should find and correct the cause of the problem.

Table 4 lists all API return codes by type. You should refer to this table when programming return code handling in your application. Following this table is an alphabetic listing of the API return codes, together with explanations and recommended program actions.

The return codes defined for the Text Search Engine library service interfaces are listed and explained in “Error handling concept and return codes” on page 185.

Table 4. API return codes, listed by type

Type	Return codes
No error	RC_CONFLICTING_TASK_RUNNING RC_CONTINUATION_MODE_ENTERED RC_DICTIONARY_NOT_FOUND RC_DONE RC_EMPTY_INDEX RC_EMPTY_LIST RC_EMPTY_QUERY RC_END_OF_INFORMATION RC_FUNCTION_DISABLED RC_INDEX_DELETED RC_INDEX_SUSPENDED RC_MORE_INFORMATION RC_NO_ACTION_TAKEN RC_PROCESSING_LIMIT_EXCEEDED RC_SERVER_BUSY RC_STOPWORD_IGNORED

Table 4. API return codes, listed by type (continued)

Type	Return codes
User error	RC_CAPACITY_LIMIT_EXCEEDED RC_CCS_NOT_SUPPORTED RC_CONFLICT_WITH_INDEX_TYPE RC_DATASTREAM_SYNTAX_ERROR RC_DOCMOD_READ_PROBLEM RC_DOCUMENT_MODEL_ALREADY_EXISTS RC_DOCUMENT_CURR_NOT_ACCESSIBLE RC_DOCUMENT_GROUP_NOT_FOUND RC_DOCUMENT_IN_ERROR RC_DOCUMENT_NOT_FOUND RC_DOCUMENT_NOT_IN_VIEW RC_DOCUMENT_NOT_SUPPORTED RC_DOCUMENT_NOT_TO_INDEX RC_FUNCTION_NOT_SUPPORTED RC_INCORRECT_AUTHENTICATION RC_INCORRECT_HANDLE RC_INCORRECT_INDEX_NAME RC_INCORRECT_LIBRARY_ID RC_INCORRECT_LOCATION RC_INCORRECT_LS_EXECUTABLES RC_INCORRECT_RELEVANCE_VALUE RC_INVALID_ATTRIBUTE_VALUE RC_INVALID_IDENTIFIER RC_INVALID_SECTION_TYPE
User error (continued)	RC_INDEX_ALREADY_EXISTS RC_INDEX_ALREADY_OPENED RC_INDEX_NOT_MEMBER_OF_GROUP RC_INDEX_NOT_OPEN RC_INVALID_MASKING_SYMBOL RC_LANGUAGE_NOT_SUPPORTED RC_LOCATION_IN_USE RC_MAX_INPUT_SIZE_EXCEEDED RC_MEMBER_OF_INDEX_GROUP RC_MISSING_DEFAULT_MODEL RC_NO_RANKING_DATA_AVAILABLE RC_QUERY_SCOPE_TOO_COMPLEX RC_REQUEST_IN_PROGRESS RC_RESULT_ALREADY_RANKED RC_RESULT_VIEW_EXISTS RC_SECTION_NAME_ALREADY_EXISTS RC_SECTION_TAG_ALREADY_EXISTS RC_SERVER_VERSION_NOT_CURRENT RC_THESAURUS_PROBLEM RC_UNKNOWN_COMMUNICATION_TYPE RC_UNKNOWN_CONDITION RC_UNKNOWN_DOCUMENT_MODEL_NAME RC_UNKNOWN_INDEX_NAME RC_UNKNOWN_INDEX_TYPE RC_UNKNOWN_SECTION_NAME RC_UNKNOWN_SERVER_INFORMATION RC_UNKNOWN_SERVER_NAME RC_UNKNOWN_SESSION_POINTER
Internal program error	RC_UNEXPECTED_ERROR

Table 4. API return codes, listed by type (continued)

Type	Return codes
Resource constraints	RC_BROWSER_TIME_OUT RC_CAPACITY_LIMIT_EXCEEDED RC_MAX_NUMBER_OF_BUSY_INDEXES RC_MAX_NUMBER_OF_INDEXES RC_MAX_NUMBER_OF_OPEN_INDEXES RC_MAX_NUMBER_OF_RESULTS RC_MAX_NUMBER_OF_TASKS RC_MAX_OUTPUT_SIZE_EXCEEDED RC_NOT_ENOUGH_MEMORY RC_QUERY_TOO_COMPLEX
Other errors	RC_COMMUNICATION_PROBLEM RC_FUNCTION_IN_ERROR RC_INDEX_GROUP_SEARCH_ERROR RC_INDEX_NOT_ACCESSIBLE RC_INDEX_SPECIFIC_ERROR RC_INSTALLATION_PROBLEM RC_IO_PROBLEM RC_LINGUISTIC_SERVICE_FAILED RC_LS_FUNCTION_FAILED RC_LS_NOT_EXECUTABLE RC_NO_RAT_EXPANSION RC_SERVER_CONNECTION_LOST RC_SERVER_IN_ERROR RC_SERVER_NOT_AVAILABLE RC_STOPWORDLIST_NOT_ACCESSIBLE RC_WRITE_TO_DISK_ERROR

Table 5. API return codes, listed by number

Return codes	Values
RC_DONE	0
RC_CONTINUATION_MODE_ENTERED	1
RC_END_OF_INFORMATION	2
RC_EMPTY_LIST	3
RC_MORE_INFORMATION	4
RC_INDEX_GROUP_SEARCH_ERROR	7
RC_INDEX_SPECIFIC_ERROR	8
RC_DICTIONARY_NOT_FOUND	9
RC_PROCESSING_LIMIT_EXCEEDED	12
RC_UNKNOWN_SERVER_NAME	16
RC_INCORRECT_AUTHENTICATION	17
RC_DATASTREAM_SYNTAX_ERROR	18
RC_QUERY_SCOPE_TOO_COMPLEX	20
RC_QUERY_TOO_COMPLEX	22
RC_MEMBER_OF_INDEX_GROUP	23
RC_UNKNOWN_INDEX_NAME	24
RC_INCORRECT_HANDLE	25
RC_INDEX_NOT_MEMBER_OF_GROUP	26
RC_UNKNOWN_SESSION_POINTER	27
RC_UNKNOWN_COMMUNICATION_TYPE	29
RC_UNKNOWN_SERVER_INFORMATION	30
RC_INVALID_MASKING_SYMBOL	31
RC_UNEXPECTED_ERROR	32
RC_SERVER_NOT_AVAILABLE	33
RC_INDEX_ALREADY_OPENED	35
RC_MAX_NUMBER_OF_OPEN_INDEXES	36
RC_MAX_NUMBER_OF_RESULTS	37

Table 5. API return codes, listed by number (continued)

Return codes	Values
RC_CCS_NOT_SUPPORTED	41
RC_LANGUAGE_NOT_SUPPORTED	42
RC_CONFLICT_WITH_INDEX_TYPE	43
RC_MAX_INPUT_SIZE_EXCEEDED	46
RC_SERVER_BUSY	47
RC_SERVER_CONNECTION_LOST	48
RC_SERVER_IN_ERROR	49
RC_REQUEST_IN_PROGRESS	50
RC_UNKNOWN_INDEX_TYPE	51
RC_INCORRECT_INDEX_NAME	52
RC_INCORRECT_LS_EXECUTABLES	53
RC_INCORRECT_LIBRARY_ID	54
RC_INDEX_ALREADY_EXISTS	55
RC_MAX_NUMBER_OF_INDEXES	56
RC_INCORRECT_LOCATION	57
RC_LOCATION_IN_USE	58
RC_UNKNOWN_CONDITION	59
RC_INDEX_DELETED	60
RC_INDEX_SUSPENDED	61
RC_INDEX_NOT_ACCESSIBLE	62
RC_MAX_NUMBER_OF_BUSY_INDEXES	63
RC_CONFLICTING_TASK_RUNNING	64
RC_NOT_ENOUGH_MEMORY	65
RC_MAX_OUTPUT_SIZE_EXCEEDED	68
RC_COMMUNICATION_PROBLEM	70
RC_NO_ACTION_TAKEN	71
RC_EMPTY_INDEX	72
RC_EMPTY_QUERY	73
RC_INSTALLATION_PROBLEM	74
RC_FUNCTION_DISABLED	75
RC_FUNCTION_IN_ERROR	76
RC_IO_PROBLEM	77
RC_WRITE_TO_DISK_ERROR	78
RC_SERVER_VERSION_NOT_CURRENT	79
RC_FUNCTION_NOT_SUPPORTED	80
RC_RESULT_ALREADY_RANKED	81
RC_RESULT_VIEW_EXISTS	82
RC_INDEX_NOT_OPEN	83
RC_NO_RANKING_DATA_AVAILABLE	84
RC_LINGUISTIC_SERVICE_FAILED	85
RC_THESAURUS_PROBLEM	86
RC_INVALID_IDENTIFIER	88
RC_DOCUMENT_MODEL_ALREADY_EXISTS	89
RC_UNKNOWN_DOCUMENT_SECTION_NAME	90
RC_DOCMOD_READ_PROBLEM	91
RC_UNKNOWN_DOCUMENT_MODEL_NAME	92
RC_SECTION_NAME_ALREADY_EXISTS	94
RC_SECTION_TAG_ALREADY_EXISTS	95
RC_MAX_NUMBER_OF_TASKS	96
RC_LS_NOT_EXECUTABLE	97
RC_LS_FUNCTION_FAILED	98
RC_CAPACITY_LIMIT_EXCEEDED	99

Table 5. API return codes, listed by number (continued)

Return codes	Values
RC_DOCUMENT_NOT_ACCESSIBLE	100
RC_DOCUMENT_CURR_NOT_ACCESSIBLE	101
RC_DOCUMENT_NOT_TO_INDEX	102
RC_DOCUMENT_NOT_FOUND	103
RC_DOCUMENT_IN_ERROR	104
RC_DOCUMENT_NOT_SUPPORTED	105
RC_CROSSIDX_SEARCH_NOT_ALLOWED	110
RC_DOCUMENT_GROUP_NOT FOUND	111
RC_INVALID_ATTRIBUTE_VALUE	112
RC_INVALID_SECTION_TYPE	113
RC_INCORRECT_RELEVANCE_VALUE	120
RC_NO_RAT_EXPANSION	130
RC_DOCUMENT_NOT_IN_VIEW	131
	200
	201
	202
RC_STOPWORDLIST_NOT_ACCESSIBLE	203

RC_CCS_NOT_SUPPORTED

Explanation:

For EhwSearch: A CCSID for the index (CCSID) specified in the *query* parameter is not supported by Text Search Engine.

For EhwCreateIndex: No CCSID for the index (XCCSID) is specified for an Ngram index, or the specified CCSID is not supported on the server platform.

Programmer response: Adapt your application so that it does not specify unsupported CCSIDs. All CCSIDs supported by Text Search Engine are defined in file IMOLANG.H provided with Text Search Engine.

With an Ngram index, use only CCSIDs supported by the platform you are using.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_COMMUNICATION_PROBLEM

Explanation: Communication with the Text Search Engine server failed.

Problem determination: The error could be caused by a lack of storage space or by an incorrect installation of Text Search Engine.

Programmer response: Issue an EhwEndSession call and end the application, saving any information that can help to find the cause of the error. For problem determination purposes, you can continue to issue any Text Search Engine API calls.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_CONFLICTING_TASK_RUNNING

Explanation: The request cannot be processed by Text Search Engine at this time. There is a server task running that updates or reorganizes the information-retrieval index. No EhwUpdateIndex or EhwReorgIndex calls are accepted while this task is active.

Programmer response: You can issue the same call (EhwUpdateIndex or EhwReorgIndex) again later.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_CONFLICT_WITH_INDEX_TYPE

Explanation:

For EhwSearch: A search term of the *query* parameter contains one of the following:

- One of the precision qualities BOUND, CSENS, or ESTEM, when the index is not Ngram, or, in the case of CSENS, when the index is not created with the XTADD_CASE_ENABLE option.

For EhwCreateIndex: An additional feature that is not supported for the chosen index type is specified, or a CCSID for the index (XCCS) is specified although the index type is not Ngram.

For EhwCreateIndexGroup: An index added to the group differs from others in its base type.

For EhwSetIndexingRules: A default format not supporting section recognition has been chosen for an index with property XTPROP_SECTIONS_ENABLED.

Programmer response: Adapt your application to prevent the specification of query options that conflict with the index type.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_CONTINUATION_MODE_ENTERED

Explanation: The requested information is too large for the buffers. The information is therefore returned in separate blocks. This code indicates that a block of data is returned and that there is more information available.

Programmer response: If you want to keep the information provided, you must save it before you ask for the next block of data. To get the next block of information, issue the same call again.

Any other API function call cancels the continuation mode. If you do not want to continue, you can explicitly cancel the continuation mode with an EhwCancelContinuation call.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_DATASTREAM_SYNTAX_ERROR

Explanation: An input parameter provided in data stream format contains at least one syntactic error.

Programmer response: Issue an EhwEndSession call and end the application, saving any information that can help to find the cause of the error.

Diagnosis information: For EhwSearch calls, the *diagnosis information* parameter contains an offset value that can help you determine the cause of the error. The value points to a data stream item of the *query* parameter that caused Text Search Engine to stop further parsing. For example, the item may be out of sequence or may have an unknown identifier or type, or it may contain an invalid value.

RC_DICTIONARY_NOT_FOUND

Explanation: Text Search Engine linguistic services cannot find the proper dictionary files. The query is processed without linguistic support.

Problem determination: This could be caused by an

installation problem. The dictionary files corresponding to the specified language code(s) are not available on the resourcs path specified during client or server installation or configuration.

Programmer response: You can continue to issue any API calls.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_DOCMOD_READ_PROBLEM

Explanation: *For EhwCreateIndex:* The global document model definition file which should reside in the server instance directory could not be read.

For EhwSearch and EhwSetIndexingRules: The document model definition file which should reside in the data directory of the index, could not be read.

Read failures could be due to:

- A missing or inaccessible file
- Invalid content (a model is declared but not specified in the file)
- An entry specifying a model is invalid; duplicate "author =" entries, for example.
- When nesting sections, a level in the hierarchy is skipped; example:

```
book   = book
author = book/title/author
```

- Nesting encountered for section of type other than text

Programmer response: Make sure the requested model definition file is accessible to the search engine.

RC_DOCUMENT_CURR_NOT_ACCESSIBLE

Explanation: The document is found but cannot be opened.

Problem determination: This could occur if the document is accessed exclusively by another task or another user.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_DOCUMENT_GROUP_NOT_FOUND

Explanation: The library service call LIB_list_documents cannot access the requested document group.

Problem determination: This could occur if the document group (such as a files ystem) is not available to the application.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_DOCUMENT_IN_ERROR

Explanation: See “Error handling concept and return codes” on page 185.

RC_DOCUMENT_MODEL_ALREADY_EXISTS

Explanation: This can occur when creating a model which has already been defined in the target document model file.

Problem determination: Use a different name for the new model to be defined.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_DOCUMENT_NOT_FOUND

Explanation: See “Error handling concept and return codes” on page 185.

RC_DOCUMENT_NOT_SUPPORTED

Explanation: The document is found and can be opened, but is not of a supported type.

Programmer response: For a list of the document formats that can be indexed, refer to the IMOLSDEF.H header file in the library services toolkit.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_DONE

Explanation: The function completed successfully. If there is an output buffer, it contains all information requested or, if the previous call returned RC_CONTINUATION_MODE_ENTERED, the last block of the requested information.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_EMPTY_INDEX

Explanation: The information-retrieval index specified in the search request is empty. Either no documents have ever been added to this index or all documents have been removed from it. The returned result handle is not valid.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_EMPTY_LIST

Explanation: There is no information to be returned for the current request:

- If it was an EhwListServers call, there is no Text Search Engine server that can be connected to the client workstation.

- If it was an EhwListIndexes call, no information-retrieval index is defined on the information-retrieval server.
- If it was any call targeted at a search request, no documents were found by the search.
- If it was any other call returning a list, the list contains no entries.

Problem determination: This may be due to one of the following:
- No Text Search Engine servers have been defined for use with the client workstation.
- The Text Search Engine server has not been installed correctly.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_EMPTY_QUERY

Explanation: The specified *query* parameter was analyzed and processed linguistically by Text Search Engine. All search terms in the query contained only stop words (words not indexed by Text Search Engine) that were removed from the query. The result was an empty search request.

Programmer response: Inform the users of your application that stop words should be avoided in Text Search Engine queries, except when specified within a phrase.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_FUNCTION_DISABLED

Explanation: Use of the requested function has been prevented by the administrator. The API call cannot be processed.

Programmer response: You can continue to issue any API calls. Use of the function can be permitted again by the administrator.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_FUNCTION_IN_ERROR

Explanation: The requested function has been locked due to an error situation on the Text Search Engine server. The API call cannot be processed.

Programmer response: Issue an EhwEndSession call and end the application, saving any information that can help to find the cause of the error. For problem determination purposes, you can continue to issue any calls to the Text Search Engine API services. The administrator can unlock the function using the Text Search Engine Administration dialog.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_GTR_ERROR

Explanation: The engine working on Ngram indexes failed with an unexpected error.

Programmer response: Issue an EhwEndSession call and end the application, saving any information that can help to find the cause of the error. For problem determination purposes, you can continue to issue any calls to the Text Search Engine API services.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_INCORRECT_AUTHENTICATION

Explanation: 1. You started a session that used an administration function in which the user ID or the password was incorrect, or

2. You started an administration function in a session in which the user ID or password has not been provided for a user in the *smadmin* group on the machine where Text Search Engine is installed.

Programmer response: 1. If you used an incorrect user ID or password, issue an EhwStartSession call again, supplying the correct user ID and password.

2. If you used an administration function in a session that was started without supplying a user ID or password, close this session and open a new one supplying the user ID and password of a user belonging to the *smadmin* user group on the machine on which Text Search Engine is installed.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_INCORRECT_HANDLE

Explanation: A handle specified in an input parameter is not valid. It must be a handle that was returned by a previous call that is not obsolete.

Programmer response: Issue an EhwEndSession call and end the application, saving any information that can help to find the cause of the error.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_INCORRECT_INDEX_NAME

Explanation: The *index name* specified in the *session information* parameter of EhwCreateIndex did not comply with the following:
- The maximum length is restricted to eight.
- The characters have to be uppercase letters A-Z or digits 0-9 of the coded characters set 00819 (ISO 8859-1).

The current request cannot be processed.

Programmer response: Issue an EhwEndSession call

and end the application, saving any information that can help to find the cause of the error.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_INCORRECT_LIBRARY_ID

Explanation: The *library identifier* specified in the *session information* parameter of EhwCreateIndex exceeded the maximum length of 250.

Programmer response: Issue an EhwEndSession call and end the application, saving any information that can help to find the cause of the error.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_INCORRECT_LOCATION

Explanation: The *index* or *work location* specified in the *session information* parameter of EhwCreateIndex could not be accessed or created due to one of the following:

- The specified drive does not exist.
- The directory name is equal to the name of an existing file.
- The path name does not comply to the naming conventions of the server system (UNC or 8.3).

The current request cannot be processed.

Programmer response: Issue an EhwEndSession call and end the application, saving any information that can help to find the cause of the error.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_INCORRECT_LS_EXECUTABLES

Explanation: An *executables name* specified in the *session information* parameter of EhwCreateIndex did not comply with the following:

- The maximum length is restricted to eight.
- The characters have to be uppercase letters A-Z or digits 0-9 of the coded characters set 00819 (ISO 8859-1).

The current request cannot be processed.

Programmer response: Issue an EhwEndSession call and end the application, saving any information that can help to find the cause of the error.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_INDEX_ALREADY_EXISTS

Explanation: Text Search Engine received a call to create an information-retrieval index that was already existing. The EhwCreateIndex call cannot be processed.

Programmer response: If you intend to replace an existing index, first delete it using the EhwDeleteResult call before attempting to create one of the same name.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_INDEX_ALREADY_OPENED

Explanation: A call to open an information-retrieval index was received while the requested index was already opened from the current session. An information-retrieval index can only be opened once per API session.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_INDEX_DELETED

Explanation: Text Search Engine received a request relating to an information-retrieval index that was deleted from another session. The current API call cannot be processed.

Programmer response: If the index is currently opened, close it using the EhwCloseIndex call. You can continue to issue any calls to Text Search Engine API services.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_INDEX_GROUP_SEARCH_ERROR

Explanation: An RC_INDEX_GROUP_SEARCH_ERROR is returned when, during a cross-index search, all indexes had problems. There will be a search result handle, but no document IDs in the result. The result handle can be used only to obtain the return code for each index of the group being searched on. Issue an EhwGetProblemInfo call, which returns one return code for each index in error.

Programmer response: Check the index-specific return codes.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_INDEX_NOT_ACCESSIBLE

Explanation: Text Search Engine received a request relating to an information-retrieval index that could not be accessed. The current API call cannot be processed.

Programmer response: Issue an EhwEndSession call and end the application, saving any information that can help to find the cause of the error.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_INDEX_NOT_MEMBER_OF_GROUP

Explanation: An error occurred when trying to add a scope to a query for a cross-index search. An index name supplied in the input data stream does not belong to the index group represented by the input Index_Group_Handle.

Programmer response: Correct the error after having checked the data stream starting at the offset delivered in the diagnosis information. Issue a new EhwAddQueryScope call.

Diagnosis information: The diagnosis information returned with this return code supplies the offset within the input data stream, where the error was detected.

RC_INDEX_NOT_OPEN

Explanation: An index has to be open to select a result list by its name as a selection criterion.

RC_INDEX_SPECIFIC_ERROR

Explanation: This return code means that during a cross-index search an index-specific error occurred. There will be a search result, but no document IDs of the index in error. To obtain further information about index-specific errors you can issue an EhwGetProblemInfo call, which returns one return code for each index in error.

Programmer response: Check the index-specific return codes.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_INDEX_SUSPENDED

Explanation: Text Search Engine received a request relating to an information-retrieval index that was suspended from another or the current session. The current API call cannot be processed.

Programmer response: If the index is currently open, close it using the EhwCloseIndex call. You can continue to issue any calls to Text Search Engine API services, but it is not recommended to resume the index, using the EhwResumeIndex call, before checking the reason for the suspension.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_INSTALLATION_PROBLEM

Explanation: Text Search Engine has encountered an installation problem.

Problem determination: This code is returned when one of the following applies:

- The environmental variable that points to the location of the client configuration file is incorrect.
- The client instance file is missing.
- Entries in the client instance file are incorrect.

Programmer response: Issue an EhwEndSession call. Do not start a new session with the same server until the problem is corrected.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_INVALID_ATTRIBUTE_VALUE

Explanation: An invalid attribute value has been specified for EhwSearch.

Programmer response: Use a value which is valid with respect to supported formats for non-text section types. For a list of supported types and formats, see Text Search Engine: Customization and Administration, SH12-6365.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_INVALID_IDENTIFIER

Explanation: In EhwCreateDocumentModel, a section or model name contains invalid code points.

Programmer response: Make sure the rules for the spelling of section and model names are respected.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_INVALID_SECTION_TYPE

Explanation: For EhwSearch, the section list used to qualify an attribute criterion contains sections of mixed types. For EhwCreateDocumentModel, an invalid value was found for section type.

Programmer response: For EhwSearch, make sure that all sections used in a section list are of the same type. For EhwCreateDocumentModel, make sure that the datastream item is one of the supported identifiers as defined in imoapic.h.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_INVALID_MASKING_SYMBOL

Explanation: A mask definition in the *query* parameter specifies identical RCM and CSM masking symbols.

Programmer response: Adapt your application to prevent ambiguous definitions of masking symbols for search terms.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_IO_PROBLEM

Explanation: An error occurred when the Text Search Engine server attempted to open or read one of its index files or a dictionary. This can be due to one of the following:

- An unintentional action by the administrator, such as the deletion of an index file.
- Incorrect installation of Text Search Engine.

Programmer response: Issue an EhwEndSession call and end the application, saving any information that can help to find the cause of the error.

Problem determination: Check with the administrator that:

- All files of the current information-retrieval index exist
- The Text Search Engine environment variables are set correctly.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_LANGUAGE_NOT_SUPPORTED

Explanation: The *language identifier* specified in the *query* parameter is not supported by Text Search Engine.

Programmer response: Adapt your application to prevent the specification of unsupported or unknown language identifiers. The language identifiers supported by Text Search Engine are defined in file IMOLANG.H provided with Text Search Engine.

Diagnosis information: There is no diagnosis information returned for this return code.

RC_LOCATION_IN_USE

Explanation: The *index* or *work location* specified in the *session information* parameter of EhwCreateIndex were already in use by Text Search Engine. The current request cannot be processed.

Programmer response: Inform the user and issue the same call again with other location specification.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_LS_FUNCTION_FAILED

Explanation: A function from within the library services was called, but it failed to run correctly, giving a return code or invalid output parameters.

Programmer response: Check the implementation of functions in the library services for compliance to the API description. If this return code comes from Text Search Engine-provided library services, check the document that caused the problem for consistency. If there is no obvious reason for failure, save any information that can help to find the cause of the error, and report it to your IBM representative.

RC_LS_NOT_EXECUTABLE

Explanation: A library service was called, but the Text Search Engine was not able to execute it. This may be due to an installation problem.

Programmer response: Check the name of the library service as defined for the index showing the error (see "EhwCreateIndex" on page 52). Ensure that this program exists in the path used for executable libraries and that is has been built properly.

RC_MAX_INPUT_SIZE_EXCEEDED

Explanation: The value of the input parameter for datastream length exceeds the limit allowed for the API call.

Programmer response: Use the correct value for the input parameter. If necessary, adjust the size of input buffer.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_MAX_NUMBER_OF_BUSY_INDEXES

Explanation: The requested function has been prevented by the search service, because the maximum number of indexes is currently active.

Programmer response: Reissue the function call after a short period of time. In general the problem is only temporary. For configuration information, see *Text Search Engine: Customization and Administration,* SH12-6365.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_MAX_NUMBER_OF_INDEXES

Explanation: There are too many Text Search Engine indexes to be stored. The EhwCreateIndex call cannot be processed.

Programmer response: Delete indexes that are no longer required. Alternatively, you can configure the system to support more indexes. To do this, add to the `[DAEMONS]` section in the server configuration file an entry like `MaxIndexEntries = newNumber`, where `newNumber` is the new value for the maximum number of allowed indexes. (For details, see *Text Search Engine: Customization and Administration,* SH12-6365.) Stop and restart the server so that the new setting is enabled.

Diagnosis information: There is no diagnosis information returned for this return code.

RC_MAX_NUMBER_OF_OPEN_INDEXES

Explanation: There are too many index handles to be stored. The EhwOpenIndex call cannot be processed.

Programmer response: Close an open index, using the EhwCloseIndex call, before attempting to open another.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_MAX_NUMBER_OF_RESULTS

Explanation: There are too many result handles to be stored. The EhwSearch call cannot be processed.

Programmer response: Delete search results that are no longer required for browsing.

Problem determination: With the Text Search Engine API services, approximately 800 search results can be handled per session. The exact number depends on the complexity of the queries and the size of the results. You should not normally allow the maximum number of results to accumulate because it affects processing performance.

Diagnosis information: There is no diagnosis information returned for this return code.

RC_MAX_NUMBER_OF_TASKS

Explanation: The requested administration function has been rejected to run by the search service, since the maximum number of administration tasks currently is active. Administration tasks may have been started via EhwUpdateIndex or EhwReorgIndex.

The maximum number of tasks for administration functions is a subset of the maximum number of indexes allowed to be active at a time.

Programmer response: Resubmit the same function later, but keep in mind that administration tasks can run for a relatively long time. The problem usually is only temporary.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_MEMBER_OF_INDEX_GROUP

Explanation: When an EhwCloseIndex call is issued, the index you want to close must not be a member of an existing index group.

Programmer response: Delete the index group(s) the index belongs to and reissue the EhwCloseIndex call.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_MISSING_DEFAULT_MODEL

Explanation: *For EhwUpdate:* Update was requested for an index that has the property XTPROP_SECTIONS_ENABLED, but no default document model has been defined for that index.

Programmer response: Use API call EhwSetIndexingRules to define which default should be used for the document model.

RC_NO_ACTION_TAKEN

Explanation: There is no operation to be performed for the current request:

- If it was an EhwReorgIndex call, the current information-retrieval index had just been reorganized when the request was received.
- If it was an EhwUpdateIndex call, the Text Search Engine indexing queue did not contain any entries when the request was received.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_NO_RANKING_DATA_AVAILABLE

Explanation: A function call assumes that basic ranking data or rank values are given, and does not find any.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_NO_RAT_EXPANSION

Explanation: No relevance values are assigned to the documents in the result view, or all relevance values are zero. Use EhwAssignRelevance for one or more documents in the result view. Make sure that at least one document is assigned a nonzero relevance.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_NOT_ENOUGH_MEMORY

Explanation: There is a lack of storage space on the client or on the server system. The current request cannot be processed.

Programmer response: Attempt to issue an EhwEndSession call to release storage space and end the application, saving any information that can help to find the cause of the error.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_PROCESSING_LIMIT_EXCEEDED

Explanation: The current EhwSearch request exceeded the maximum processing time specified in the *query* parameter. The request was canceled, and no valid *result handle* was returned.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_QUERY_SCOPE_TOO_COMPLEX

Explanation: The scope information you wanted to add to a query for a cross index search using EhwAddQueryScope is too long (see also explanation of RC_QUERY_TOO_COMPLEX).

Programmer response: Shorten the scope information and resubmit the EhwAddQueryScope call.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_QUERY_TOO_COMPLEX

Explanation: The specified query is too complex and cannot be processed by Text Search Engine.

Programmer response: Adapt your application to prevent over-complex queries, or to split complex queries into a series of simpler ones.

Problem determination: There are no limits in the Text Search Engine API to the number of operators or the number of subqueries that can be used. When a complex query is received, the Text Search Engine API tries to divide it into several query blocks to be sent to the Text Search Engine server. However, it can reach a stage where this can no longer be done, and this return code is returned.

This error return code is also returned when excessive use of masking symbols or excessive use of the SYN option expand the original query to a size that cannot be managed by Text Search Engine.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_REQUEST_IN_PROGRESS

Explanation: A Text Search Engine API service was called while another API request was active for the same API session.

Programmer response: Issue an EhwEndSession call and end the application, saving any information that can help to find the cause of the error.

Problem determination: The Text Search Engine API does not support concurrent access to the same API session.

All applications running concurrently in the same process should handle their own API sessions.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_RESULT_ALREADY_RANKED

Explanation: The result object has already been assigned rank values. Ranking it twice is declined.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_RESULT_VIEW_EXISTS

Explanation: At least one result view may be created from each result object. This return code indicates that a second attempt has been made.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_SECTION_NAME_ALREADY_EXISTS

Explanation: EhwCreateDocumentModel encountered a duplicate for a section name specified in the datastream.

Programmer response: Ensure that naming rules are followed by the application. These require:
- Uniqueness of model name in any document model file
- Uniqueness of section name within any model definition
- Uniqueness of section tag within any model definition

Diagnosis information: There is no diagnosis information returned with this return code.

RC_SECTION_TAG_ALREADY_EXISTS

Explanation: EhwCreateDocumentModel encountered a duplicate for a section tag specified in the datastream.

Programmer response: Ensure that the naming rules are followed by the application. For a list of these rules, see RC_SECTION_NAME_ALREADY_EXISTS.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_SERVER_BUSY

Explanation: The Text Search Engine client cannot currently establish a session with the requested Text Search Engine server, or the Text Search Engine server communication link was interrupted and cannot be re-established.

Problem determination: The Text Search Engine server had been started correctly but the maximum number of parallel server processes was reached. Unless this was a temporary problem, the communication configuration on the Text Search Engine server should be adapted.

Programmer response: End the application and ask the user to start it again to check if this was a temporary problem.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_SERVER_CONNECTION_LOST

Explanation: The communication between client and server was interrupted and cannot be re-established.

Problem determination: The Text Search Engine server task may have been stopped by an administrator or the server workstation may have been shut down.

Programmer response: Issue an EhwEndSession call and end the application, saving any information that can help to find the cause of the error. If you want to continue processing for problem determination purposes, first issue an EhwEndSession call, then an EhwStartSession call to establish a new session with the Text Search Engine server.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_SERVER_IN_ERROR

Explanation: An error occurred on the Text Search Engine server workstation that cannot be recovered by Text Search Engine API calls.

Programmer response: End the application, saving any information that can help to find the cause of the error.

Problem determination: You can continue to issue any Text Search Engine API calls after the Text Search Engine server system has been re-booted.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_SERVER_NOT_AVAILABLE

Explanation: The Text Search Engine API services could not establish a session with the requested Text Search Engine server.

Problem determination: Probably the Text Search Engine server has not yet been started. If the Text Search Engine server has been activated correctly and the error persists, there is an installation problem.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_STOPWORD_IGNORED

Explanation: The specified query contained at least one search term consisting only of stop words. The search term was ignored when processing the query.

Programmer response: You can continue to issue any API calls. You may want to inform the users of your application that stop words should be avoided when formulating Text Search Engine queries.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_STOPWORDLIST_NOT_ACCESSIBLE

Explanation: The stopword list file cannot be found in the expected path. Stopword filtering is stopped. No dcouments are processed.

Programmer response: Check whether the stopword file corresponding to the specified language code or codes exists in the resource path specified during client installation or configuration, and whether it is readable. This path is set to the subdirectory dict in the installation directory of Text Search Engine.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_THESAURUS_PROBLEM

Explanation: In a call to EhwSearch requesting Thesaurus expansion, the relation name specified was unknown to the thesaurus used.

Programmer response: Ensure that the thesaurus filename denotes a thesaurus for which the relation name has been defined, or check the spelling of the relation name.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_UNEXPECTED_ERROR

Explanation: An error occurred that could be caused by an incorrect routine in Text Search Engine or by incorrect installation of Text Search Engine.

Programmer response: End the application, saving any information that can help to find the cause of the error. If the error is not caused by an installation problem, report it to your IBM representative.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_UNKNOWN_COMMUNICATION_TYPE

Explanation: Text Search Engine does not know the *communication type* specified in the *session information* parameter.

Programmer response: End the application, saving any information that can help to find the cause of the error.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_UNKNOWN_CONDITION

Explanation: The *suspend condition* specified in the *processing condition* parameter is not supported by Text Search Engine.

Programmer response: Adapt your application to prevent the specification of unknown conditions. The suspend conditions supported by Text Search Engine are defined in file IMOAPIC.H provided with Text Search Engine.

Diagnosis information: There is no diagnosis information returned for this return code.

RC_UNKNOWN_DOCUMENT_MODEL_NAME

Explanation: *For EhwSetIndexingRules:* The default for document model was not found in the model definition file of the index that the command was issued for.

For *EhwGetDocumentModel* and *EhwDeleteDocumentModel*, the model name entered was not a valid name in the document model file accessed by the call.

Programmer response: Check the spelling of the requested default name for the document model to ensure that it corresponds to an entry in the document model definition file of this index.

Note: If the index already contains documents, you must not alter the content of the definition file.

RC_UNKNOWN_INDEX_NAME

Explanation: Text Search Engine does not know the specified *index name*.

Programmer response: Check the index name against the list returned by EhwListIndexes.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_UNKNOWN_INDEX_TYPE

Explanation: Text Search Engine does not know the *index type* specified in the *session information* parameter of EhwCreateIndex. The current request cannot be processed.

Programmer response: Check the index type against the list of types in imoapic.h.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_UNKNOWN_SECTION_NAME

Explanation: *For EhwSearch:* Either the input of a name for a document model was not found in the model definition file of the index, or none of the section names is defined for that model.

Programmer response: Check the spelling of the requested document model name and the section names in the following list. Check the content of the document model definition file to ensure that the requested input is valid for that index.

Note: Do not alter the content of the definition file.

RC_UNKNOWN_SERVER_INFORMATION

Explanation: Text Search Engine does not know the *server identifier* or the *application identifier* specified in the *session information* parameter.

Programmer response: End the application, saving any information that can help to find the cause of the error.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_UNKNOWN_SERVER_NAME

Explanation: Text Search Engine does not know the *server name* specified in the *session information* parameter.

Programmer response: Check the server name against the list returned by EhwListServers.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_UNKNOWN_SESSION_POINTER

Explanation: The session pointer specified in the current service call is incorrect or obsolete.

Programmer response: End the application, saving any information that can help to find the cause of the error.

Problem determination: Note that a session pointer is obsolete after an EhwEndSession call.

Diagnosis information: There is no diagnosis information returned with this return code.

RC_WRITE_TO_DISK_ERROR

Explanation: A write error occurred that could be caused by a full disk on the Text Search Engine server workstation or by incorrect installation of Text Search Engine.

Programmer response: Issue an EhwEndSession call and end the application, saving any information that can help to find the cause of the error.

Problem determination: Check with the administrator that there is enough disk space available.

Diagnosis information: There is no diagnosis information returned with this return code.

Appendix B. Error codes returned by GetIndexingMsgs and GetIndexFunctionStatus

Indexing messages occur when, for example:

- Documents cannot be indexed
- Documents are indexed, but a problem occurs
- A language dictionary cannot be found.

This appendix explains the reason codes returned by an EhwGetIndexingMsgs call or by an EhwGetIndexFunctionStatus call. If you need more information on the cause of the error, look in the log file `imodiag.log` in the server instance path. In many cases, there will be an entry for the message code telling you which index file caused a problem during update.

1 Out of storage. The server ran out of memory. Reduce the workload.

116
Datastream syntax error

280
The document has not been indexed. One of the index files could not be opened.

281
The document has not been indexed. One of the index files could not be read.

441
The document has not been indexed. This message occurs for Ngram indexes only. The document's codepage is different from the one the index was created with. This may happen for HTML and XML documents if the index was not created in UTF8.

500
The document has not been indexed. The Library Services could not be loaded. Check that the DLL is available and that the resource path is valid.

501
The document has not been indexed. Lib_Init in Library Services failed On Flat File systems: DIT file not found or not on a valid directory, or DIT contents not correct.

502
The document has not been indexed. An error has occurred while reading the document content in library service LIB_read_doc_content.

503
The document has not been indexed. An error occurred in library service LIB_access_doc.

504
The document has not been indexed. The library service LIB_doc_index_status returned an error.

505
Close document failed. The library service LIB_close_doc returned an error.

506
End Library Services failed The library service LIB_end returned an error.

507
The library service call LIB_read_doc_content failed with an unexpected return code.

508
The library service call LIB_close_doc returned a RC_TERMINATION error.

545
The document has not been indexed. One of the temporary index files could not be opened.

546
The document has not been indexed. One of the temporary index files could not be closed.

548
Internal error. Send the information in the diagnosis log to your IBM representative.

549, 550
Out of storage (alloc failed). The server ran out of memory. Reduce the workload.

551-564
The document has not been indexed. One of the index files could not be opened, read, written to or closed. Check that there is enough space on the disk used for the index and that access rights are correct.

565
Out of storage (alloc failed). The server ran out of memory. Reduce the workload.

566-587
The document has not been indexed. One of the index files could not be opened, read, written to or closed.

588-590
Internal error. Send the information in the diagnosis log to your IBM representative.

591-604
The document has not been indexed. One of the index files could not be opened, read, written to or closed.

605
Out of storage (alloc failed). The server ran out of memory. Reduce the workload.

606-623
The document has not been indexed. One of the index files could not be opened, read, written to or closed. Check that there is enough space on the disk used for the index and that access rights are correct.

624
Out of storage (alloc failed). The server ran out of memory. Reduce the workload.

625-631
The document has not been indexed. One of the index files could not be opened, read, written to or closed. Check that there is enough space on the disk used for the index and that access rights are correct.

632
One of the temporary files created during indexing could not be opened with write access. Check the access rights.

633
One of the temporary files created during indexing could not be closed.

634
One of the temporary files created during indexing could not be written. Check that the index working directory has enough disk space.

635
One of the temporary files created during indexing could not be read.

636
One of the temporary files created during indexing could not be opened with read access. Check the access rights.

659
One of the temporary files created during indexing could not be opened.

660
One of the temporary files created during indexing could not be written.

661
One of the temporary files created during indexing could not be closed.

662
One of the temporary files created during indexing could not be opened.

663
One of the temporary files created during indexing could not be written.

664
One of the temporary files created during indexing could not be closed.

665
One of the temporary files created during indexing could not be opened.

667
One of the temporary files created during indexing could not be written.

668-669
The document was not indexed. There was a matching problem with section tags encountered in the document versus those defined in document models file.

670-672
The document has not been indexed. One of the index files could not be opened, read, written to or closed. Check that there is enough space on the disk used for the index and that access rights are correct.

673
Out of storage (alloc failed). The server ran out of memory. Reduce the workload.

674
Internal error, send the information in the diagnosis log to your IBM representative.

675-687
The document has not been indexed. One of the index files could not be opened, read, written to or closed. Check that there is enough space on the disk used for the index and that access rights are correct.

688, 690
Out of storage (alloc failed). The server ran out of memory. Reduce the workload. Try smaller values in the configuration file.

689
Internal error. Send the information in the diagnosis log to your IBM representative.

691-695
The document has not been indexed. One of the index files could not be opened, read, written to or closed. Check that there is enough space on the disk used for the index and that access rights are correct.

696-707
Internal error, send the information in the diagnosis log to your IBM representative.

708
Out of storage (alloc failed). The server ran out of memory. Reduce the workload. Try smaller values in configuration file.

709-718
The document has not been indexed. One of the index files could not be opened, read, written to or closed. Check that there is enough space on the disk used for the index and that access rights are correct.

719-721
Internal error, send the information in the diagnosis log to your IBM representative.

722, 729
Out of storage (alloc failed). The server ran out of memory. Reduce the workload. Try smaller values in configuration file. For more information see *Text Search Engine: Customization and Administration,* SH12-6365.

730, 732, 733, 735-738
The document has not been indexed. One of the index files could not be opened, read, written to or closed. Check that there is enough space on the disk for the index and that access rights are correct.

731, 739-742, 744-746, 749, 755-758, 760-761, 767
Internal error, send the information in the diagnosis log to your IBM representative.

743, 748, 750-754, 759, 765-766, 768-770
The document has not been indexed. One of the index files could not be opened, read, written to or closed. Check that there is enough space on the disk used for the index and that access rights are correct.

747, 763, 764
Out of storage (alloc failed). The server ran out of memory. Reduce the workload. Try smaller values in configuration file.

815

There are two possible causes:

- One of the resource files needed to support the language used for the document causing the failure was not found
- The language requested by that document is not supported by Text Search Engine

831
The document has not been indexed. No text has been found. The document length is 0 bytes.

860
File open error. Either some dictionaries or the thesaurus files could not be found. Check your resource path for the dictionary files. The resource path is defined in the server configuration file `imosrv.ini` and in the client configuration file `imocl.ini`. If you have specified path information for your thesaurus files during search, check the location and file name.

861
The document has not been indexed. Tokenization of its text found no valid tokens. Check the document's content for validity with respect to supported languages and contained words. This error can be caused by a document containing only stopwords.

954-956
Internal error. Send the information in the diagnosis log to your IBM representative.

957-967
The document has not been indexed. One of the index files could not be opened, read, written to or closed. Check that there is enough space on the disk used for the index and that access rights are correct.

1000
An error occurred during file open. Check access rights.

1001
An error occurred during file append. Check access rights.

1002
An error occurred during file read. Maybe the file is corrupted.

1003
An error occurred during file write. Check disk space and access rights.

1005
An error occurred during file read (positioning within file). Maybe the file is corrupted.

1006
An error occurred during rename of temporary file. Check the access rights.

1007
An error occurred during file create. Check access rights.

1008
An error occurred during file compression. Check the access rights.

1009
An error occurred during file close. Maybe the file is corrupted.

1010
The specified index name is already in use. Use another index name.

1011
The specified path is already in use. Use another location.

1012
The same path is used for data and working directory. Use another location.

1013
The specified index name is invalid. Index names must be uppercase or digits and not longer than 8 characters.

1014
An error occurred during file copy. Check access rights and disk space.

1017
The index name is unknown. Check correct spelling.

1019
An error occurred during file deletion. Check access rights. This error message can occur as a ″secondary error″ - look up the diagnosis file to see if a preceding error entry gives more information.

1020
General file error. Check access rights.

1070-1074
The document has not been indexed. The codepage specified is either invalid generally, or is invalid for the index being accessed.

1085
The document has not been indexed. An error occurred when reading the index queue.

1086
The document has not been indexed. The index queue is empty.

1116-1117
The document has not been indexed. Information from the server instance initialization file could not be processed. Make sure entries in the initialization file are valid, and that the file is accessible to the application.

1129
No document has been indexed. Starting the background processing failed.

1158
An error occurred while renaming a file. Check access rights and disk space.

1162
The index files of an Ngram index may be corrupted.

1163, 1164
The document has not been indexed due to an unexpected error.

1165
The document has not been indexed due to an unexpected end-of-file condition.

1176
No more documents can be indexed for Ngram index. There is an overflow condition for document numbers (overflow of long). If there were many deletions or repeated updates of the same document, try a call to EhwReorg to solve the problem. If not, consider using a second index for new documents.

1177
The document has not been indexed. It was considered too big by the Ngram indexer.

1189
The document has not been indexed. There was a problem with boundary sequence (Korean-language specific).

1198-1200
The document has not been indexed. There was a problem with index access. The index may be corrupted.

1201
The document has not been indexed. The document codepage could not be converted to the index-specific codepage. This error is for Ngram indexes in UTF8 codepage only.

1202
The document has not been indexed. The document codepage could not be converted to the index-specific codepage due to invalid data in the document. This error is for Ngram indexes in UTF8 codepage only.

1500-1505
The document-analysis component has problems. It could either not be initialized (check LIBPATH and content of configuration file) or failed due to internal problems. See the diagnosis file for more information.

1904
The document has not been indexed. There is a problem with accessing the document model for a section-enabled index. Check access rights and for the existence of the file.

2000
The document has not been indexed. The document type is not supported. Library service Lib_access_doc returned an invalid document type.

2001
The document has not been indexed. An incorrect sequence of fields has been detected in the document's data stream.

2002
The document has not been indexed. An incorrectly structured field has been detected in the document's data stream.

2003
The document has not been indexed. Only one text section is allowed for a document in Text Search Engine text format.

2005
The document has not been indexed. A language specified in the document's data stream is not supported.

2006
The document has not been indexed. A CCSID specified in the document's data stream is not supported.

2007
The expected document format given by the library or by the default rule is not correct. The document header is incorrect for the format. Check if the default rule is a document with a special document header, and change if the rule is not correct.

2008
The document was not indexed because it could not be accessed.

2009
The document was not indexed because it was in use and could not be accessed.

2010
The document has not been indexed. The specified CCSID is not correct.

2011
The document has not been indexed because it is not a valid IBM DCA RFT or FFT document. End-of-page must be the last control in the body text of the document.

2012
The document has not been indexed because it is not a valid IBM DCA RFT or FFT document. A structured field contains an incorrect length specification.

2013
The document has not been indexed because it is not a valid IBM DCA RFT or FFT document. An incorrect control has been detected in the document.

2014
The document has not been indexed because it is not a valid IBM DCA RFT or FFT document. An incorrect multi-byte control or structured field has been detected in the document.

2015
The document has not been indexed because it is not a valid IBM DCA RFT or FFT document. Duplicate document parameters have been found.

2016
The document has not been indexed because it is not a valid IBM DCA RFT or FFT document. An empty text unit has been found.

2018
Either the document is in a format that is not supported, or there is an “exclude” entry in the DIT for the document’s extension. Check that the document has a extension that allows it to be indexed.

2020
The document has not been indexed. It is neither a WordPerfect document nor a WordPerfect file.

2021
The document has not been indexed. It is a WordPerfect file but not a WordPerfect document.

2022
The document has not been indexed. This version of WordPerfect is not supported.

2023
The document has not been indexed. It is an encrypted WordPerfect file. Store the document without encryption.

2026
An END_TXT occurred in a footnote or an endnote. Check the WordPerfect file, it may be damaged.

2028
The parser returned non document text. Check the file content, especially with respect to format-specific words. Check whether the document format is supported. If automatic format recognition fils, ensure that the correct parser is called (see “EhwSetIndexingRules” on page 144).

2030
The document has not been indexed. Either it is not a Microsoft Word file or it is a version of Word that is not supported.

2031
The document has not been indexed. Unexpected end-of-file has been detected in a Microsoft Word document.

2032
The document has not been indexed. An incorrect control has been detected in a Microsoft Word document.

2033
The document has not been indexed. It was saved in *complex* format with the *fastsave* option. Save it with the *fastsave* option off.

2034
The document has not been indexed. A required field-end mark is missing in a Microsoft Word document.

2035
The document is encrypted. Store the document in Microsoft Word without encryption.

2036
This is a Word for Macintosh document; it cannot be processed. Store the document in Word for Windows format.

2037
This Word document contains embedded OLE objects.

2040
The document has not been indexed because it is not a valid ECTF file.

2041
The document has not been indexed. It contains an .SO LEN control that is not followed by a number.

2042
The document has not been indexed. It contains an .SO LEN control that is followed by an incorrect number. The number must be between 1 and 79.

2043
The document has not been indexed. Only one .SO DOC control is allowed. Save each ECTF document in a separate file.

2044
The document has not been indexed. An .SO HDE control must be followed by begin and end tags.

2046
The document has not been indexed. The document contains text before the .SO DOC control.

2047
The document has not been indexed. The document contains text before an .SO PID control.

2048
The document has not been indexed. An end tag is missing after a begin tag.

2050
The document has not been indexed. Incorrect tags have been detected following an .SO HDE control.

2051
The document has not been indexed. End-of-line has been detected after an .SO control.

2052
The document has not been indexed. Unexpected end-of-text has been detected.

2060
The document has not been indexed. Either it is not an AmiPro document or it is a version of AmiPro that is not supported.

2061
The document has not been indexed. The length of a control in an AmiPro document is too long.

2062
The document has not been indexed. This version of AmiPro is not supported. Only AmiPro Architecture Version 4 is supported.

2063
AmiPro Style Sheets have not been indexed.

2064
The document has not been indexed. An incorrect character set has been detected. Only Lotus Character Set 82 (Windows ANSI) is supported.

2065
The document has not been indexed. Unexpected end-of-file has been detected in an AmiPro document.

2072
The document cannot be scanned because it is encrypted.

2073
The document format is inconsistent.

2074
The document has the “bad file” flag bit set.

2080
The document has not been indexed. Either it is not an RTF document or it is a version of RTF that is not supported.

2081
The document has not been indexed. An RTF control word has been detected that is too long.

2083
The document has not been indexed. Macintosh code page is not supported.

2084
The document has not been indexed. It is an RTF document, but this RTF version is not supported. Only RTF Version 1 is supported.

2090
The document has not been indexed. It is an HTML document, which contains a tag considered too long by the parser.

2093
The document has not been indexed. It is an XML document, which was rejected by the XML parser.

2100
The document is damaged or unreadable for some other reason. A new common parser could correct the problem.

2101
The document cannot be indexed because it is empty or it contains no text. Check whether the document contains only graphics.

2102
The document cannot be indexed because it is either password-protected or encrypted.

2105
The document type is known, but the filter is not available.

2106
The document cannot be indexed because it is empty.

2107
The document cannot be indexed because it cannot be opened. Check document access.

2112
The document cannot be indexed because it is an executable file.

2113
The document cannot be indexed because it is compressed.

2114
The document cannot be indexed because it is a graphic. If the graphic document format returns an acceptable piece of text, then request to include this document format in the indexing process.

2120
The output file of the user exit does not exist or is not accessible. A new common parser version could correct the problem.

2121
The output file cannot be opened for read or it is empty. A new common parser version could correct the problem.

2122
Attempting to use a user-exit output file, but no file name has been given or set in the object.

2130
The user exit program could not be run. Check if the executable can be found in the path set by the PATH environment variable. Create a trace and dump to get additional information about the environment (errno) return codes.

2131
The user exit program failed with a bad return code. Create a trace and dump to get additional information about the environment (errno) return codes.

Appendix C. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Deutschland
Informationssysteme GmbH
Department 3982
Pascalstrasse 100
70569 Stuttgart
Germany

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement or any equivalent agreement between us.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

This book is intended to help the application programmer to use the IBM Text Search Engine programming interfaces. This book documents general-use programming interface and associated guidance information for Text Search Engine.

General-use programming interfaces allow the customer to write programs that obtain the services of Text Search Engine and/or programs that provide specific library services to Text Search Engine.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States, or other countries, or both:

AIX
IBM
Intelligent Miner
MVS
z/OS

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Index

A

analysis of text
  for indexing 6
API
  error handling 191
  functions 21
  overview 1
  return codes, by type 192, 194
  what it provides 33
API functions
  EhwAddQueryScope 38
  EhwCancelContinuation 41
  EhwClearIndex 43
  EhwClearScheduledDocuments 45
  EhwCloseDocument 47
  EhwCloseIndex 48
  EhwCreateDocumentModel 49
  EhwCreateIndex 52
  EhwCreateIndexGroup 58
  EhwCreateResultView 60
  EhwDeleteDocumentModel 62
  EhwDeleteIndex 64
  EhwDeleteIndexGroup 66
  EhwDeleteResult 67
  EhwDeleteResultView 69
  EhwDelIndexingMsgs 70
  EhwEndSession 71
  EhwGetDocumentModel 72
  EhwGetIndexFunctionStatus 75
  EhwGetIndexInfo 78
  EhwGetIndexingMsgs 81
  EhwGetIndexingRules 84
  EhwGetIndexStatus 87
  EhwGetMatches 90
  EhwGetProblemInfo 94
  EhwGetResultView 96
  EhwListDocumentModels 99
  EhwListIndexes 102
  EhwListResult 104
  EhwListServers 107
  EhwOpenDocument 110
  EhwOpenIndex 113
  EhwRank 115
  EhwReorgIndex 118
  EhwResumeIndex 120
  EhwScheduleDocument 122
  EhwSearch 125
  EhwSelectResultView 139
  EhwSetIndexFunctionStatus 142
  EhwSetIndexingRules 144
  EhwSort 147
  EhwStartSession 150
  EhwSuspendIndex 154
  EhwUpdateIndex 156
  how to call 37
application
  linking 34
application *(continued)*
  set up 34
attribute list 160

B

base form, reducing terms to 7
basic text analysis 6
  for indexing terms 6
  normalization 6
  sentence recognition 7
  terms with nonalphanumeric characters 6
big-endian format 3

C

calling the API functions 37
CCSID 4, 188
character data
  passing 3
  types of 4
character masking 9
closing an index 23
compound terms, splitting 7
connecting Text Search Engine to a library 159
creating an index 22
cross-index search
  overview 26
  problem information 29
  scope of 27

D

data exchange conventions 1
data stream
  basics 2
  data passed 2
  format 2
  identifier 2
  input 35
  items 2
  length 2
  numeric fields 35
  output 35
  parameters 2
  type 2
decomposition of compound terms 7
dehyphenation 188
deleting a result 28
deleting an index 22
diagnosis information 160, 185, 191
document access 1, 160
document formats 1
document identifiers 4
document index status 160
document management system 159
document text 4

E

EhwAddQueryScope
  example 39
  format 38
  overview 27
  parameters 38
  purpose 38
  results 38
  usage 39
EhwCancelContinuation
  examples 41
  parameters 41
  purpose 41
  results 41
  usage 41
  with GetResultView 31
  with ListIndexes 22
EhwClearIndex
  examples 43
  overview 24
  parameters 43
  purpose 43
  results 43
  usage 43
EhwClearScheduledDocuments
  examples 45
  overview 24
  parameters 45
  purpose 45
  results 45
  usage 45
EhwCloseDocument
  examples 47
  parameters 47
  purpose 47
  result 47
  usage 47
EhwCloseIndex
  examples 48
  overview 23
  parameters 48
  purpose 48
  results 48
  usage 48
EhwCreateDocumentModel
  examples 51
  format 50
  parameters 49
  purpose 49
  results 49
  usage 51
EhwCreateIndex
  examples 56
  format 53
  overview 22
  parameters 52
  purpose 52
  results 52
  usage 56
EhwCreateIndexGroup
  examples 59
EhwCreateIndexGroup *(continued)*
  format 59
  overview 26
  parameters 58
  purpose 58
  results 58
  usage 59
EhwCreateResultView
  examples 60
  overview 29
  parameters 60
  purpose 60
  results 60
  usage 60
EhwDeleteDocumentModel
  examples 63
  format 62
  parameters 62
  purpose 62
  results 62
  usage 63
EhwDeleteIndex
  examples 65
  format 64
  overview 22
  parameters 64
  purpose 64
  results 64
  usage 65
EhwDeleteIndexGroup
  examples 66
  overview 26
  parameters 66
  purpose 66
  results 66
EhwDeleteResult
  examples 68
  overview 28
  parameters 67
  purpose 67
  results 67
  usage 67
EhwDeleteResultView
  examples 69
  parameters 69
  purpose 69
  results 69
EhwDelIndexingMsgs
  examples 70
  parameters 70
  purpose 70
  results 70
  usage 70
EhwEndSession
  examples 71
  overview 21
  parameters 71
  purpose 71
  results 71
  usage 71

EhwGetDocumentModel
examples 74
format 72, 73
parameters 72
purpose 72
results 72
usage 74
EhwGetIndexFunctionStatus
examples 77
format 76
overview 26
parameters 75
purpose 75
results 75
usage 76
EhwGetIndexInfo
examples 80
format 78
overview 23
parameters 78
purpose 78
results 78
usage 80
EhwGetIndexingMsgs
error codes 207
examples 82
format 81
parameters 81
purpose 81
results 81
usage 82
EhwGetIndexingRules
examples 86
format 84
overview 23
parameters 84
purpose 84
results 84
usage 85
EhwGetIndexStatus
examples 88
format 87
overview 25
parameters 87
purpose 87
results 87
usage 88
EhwGetMatches
examples 93
format 90
overview 28
parameters 90
purpose 90
results 90
usage 92
EhwGetProblemInfo
examples 95
format 94
overview 29
parameters 94
purpose 94
EhwGetProblemInfo *(continued)*
result 94
usage 95
EhwGetResultView
examples 98
format 96
overview 31
parameters 96
purpose 96
results 96
usage 97
EhwListDocumentModels
examples 100
format 100
parameters 99
purpose 99
results 99
usage 100
EhwListIndexes
examples 103
format 102
overview 22
parameters 102
purpose 102
results 102
usage 103
EhwListResult
examples 105
format 104
parameters 104
purpose 104
results 104
usage 105
EhwListServers
examples 108
format 107
overview 21
purpose 107
results 107
usage 108
EhwOpenDocument
examples 111
format 111
parameters 110
purpose 110
results 110
usage 111
EhwOpenIndex
examples 114
format 114
overview 23
parameters 113
purpose 113
results 113
usage 114
EhwRank
examples 116
overview 29
parameters 115
purpose 115
results 115

EhwRank *(continued)*
usage 115
EhwReorgIndex
examples 119
overview 25
parameters 118
purpose 118
results 118
usage 118
EhwResumeIndex
examples 121
format 121
overview 23
parameters 120
purpose 120
results 120
usage 121
EhwScheduleDocument
examples 123
format 123
overview 24
parameters 122
purpose 122
results 122
usage 123
EhwSearch
examples 137
format 126
overview 26
parameters 125
purpose 125
results 125
EhwSelectResultView
examples 140
format 140
overview 30
parameters 139
purpose 139
results 139
usage 140
EhwSetIndexFunctionStatus
examples 143
overview 26
parameters 142
purpose 142
results 142
EhwSetIndexingRules
examples 146
format 145
overview 23
parameters 144
purpose 144
results 144
usage 146
EhwSort
examples 148
format 147
parameters 147
purpose 147
results 147
usage 148
EhwStartSession
examples 152
format 150
overview 21
parameters 150
purpose 150
results 150
usage 152
EhwSuspendIndex
examples 155
format 155
overview 23
parameters 154
purpose 154
results 154
usage 155
EhwUpdateIndex
examples 157
overview 24
parameters 156
purpose 156
results 156
usage 156
end Text Search Engine session 21
environment overview 33
error codes from EhwGetIndexMsgs 207
error handling, in API 191
error handling, library services 185
external names 4

F

format
big-endian 3
little-endian 3
function prototypes 34
function status of an index 26

H

highlighting information 28

I

ID() 35
images, links to 184
IMOAPIC.H 34, 35, 191
IMOLANG.H 92
IMOLSCFS 161
IMOLSDEF.H 92, 189
IMOLSPRO.H 163
IMOLSSFS 161
index
closing 23
creating 22
cross-index search 26
deleting 22
function status 26
getting index information 23
groups 26
list of 22

index *(continued)*
opening 23
reorganizing 25
resuming 23
rules 23
setting up 21
status information 25
suspending 23
updating 24
indexing, linguistic processing for 5
input data streams 35
interfaces
API functions 37
library service functions 159
overview 1
ISO 8859-1 4

L

language of text documents
supported languages 9
LIB_access_doc 164
LIB_close_doc 166
LIB_end 169
LIB_get_doc_attr_values 170
LIB_get_doc_group_attr_values 173
LIB_init 176
LIB_list_doc_groups 178
LIB_list_documents 181
LIB_read_doc_content 184
library
connecting Text Search Engine to 159
system 1, 159
library services
attribute list function 160
document access function 160
document index status function 160
functions 160
interfaces 1
logical relationships 163
overview 159
session control function 160
specifications 163
linguistic processing
basic text analysis 6
character and word masking 9
description 5
for indexing 5
for retrieval 8
masking of characters and words 9
reducing terms to base form 7
sound expansion 9
splitting compound terms 7
stop-word filtering 7
supported languages 9
synonyms 8
thesaurus expansion 10
link library 34
linking your application 34
listing indexes 22
listing Text Search Engine servers 21
little-endian format 3

M

masking of characters and words 9
matches, getting highlight information 28
multilingual documents 1

N

natural-language text 1
normalization of terms 6
Notices 219
numerical values, passing 3

O

opening an index 23
output data streams 35

P

parameters, data stream 2
performance, improving 25
problem information for a cross-index search 29
programming hints 35

R

ranking a result view 29
recognizing sentences 7
reducing terms to base form 7
reorganizing an index 25
result
content 31
deleting 28
ranking 29
sorting 30
view, working with 29
resuming an index 23
retrieval, linguistic processing for 8
return codes
from the API 191
listed by type 192, 194
types 191
return codes (library services)
concept 185
internal program error 192
no error 191
other errors 192
resource constraints 192
user error 191
rules for indexing 23

S

SAA Coded Character Set Identifier (CCSID) 188
scheduling documents 24
scope of a cross-index search 27
search argument 132

search functions 26
search result
  content 31
  deleting 28
  ranking 29
  sorting 30
  working with views 29
search terms 4
searching multiple indexes 26
sections
  creating an index enabled for section support 31
  enabling recognition for an index 55
  in index characteristic from EhwGetIndexInfo 79
  specifying in EhwGetMatches 91
  specifying in EhwSearch 131
  specifying in the Text Search Engine text format 188
sentence recognition 7
session, starting and stopping 21
session control 160
setting up the library connection 160
setting up your application 34
sorting a result view 30
sound expansion 9
specifications for library services 163
starting a Text Search Engine session 21
stop-word filtering 7
storage considerations 35
suspending an index 23
symbolic name definitions 34
synonyms 8
system overview 33

T

term normalization 6
text analysis 6
text criterion 129
text format 187
Text Search Engine
  index 161
  names 4
  text format 1, 187
thesaurus expansion 10
types of API return codes 191

U

updating an index 24

V

VAL2() 35

W

word masking 9
WordPerfect 1

Readers' Comments — We'd Like to Hear from You

z/OS
Text Search:
Programming the Text Search Engine
Version 1.2

Publication No. SH12-6717-01

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SH12-6717-01

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Deutschland Entwicklung GmbH
Information Development, Dept. 0446
Schoenaicher Str. 220
71032 Boeblingen
Germany

Fold and Tape | **Please do not staple** | Fold and Tape

SH12-6717-01

Cut or Fold
Along Line

Program Number: 5694-A01

SH12-6717-01